Real-Time Hatching

Emil Praun, Princeton University
Hugues Hoppe, Microsoft Research
Matthew Webb, Princeton University
Adam Finkelstein, Princeton University

Abstract

Drawing surfaces using hatching strokes simultaneously conveys material, tone, and form. We present a real-time system for non-photorealistic rendering of hatching strokes over arbitrary surfaces. During an automatic preprocess, we construct a sequence of mip-mapped hatch images corresponding to different tones, collectively called a *tonal art map*. Strokes within the hatch images are scaled to attain appropriate stroke size and density at all resolutions, and are organized to maintain coherence across scales and tones. At runtime, hardware multitexturing blends the hatch images over the rendered faces to locally vary tone while maintaining both spatial and temporal coherence. To render strokes over arbitrary surfaces, we build a lapped texture parametrization where the overlapping patches align to a curvature-based direction field. We demonstrate hatching strokes over complex surfaces in a variety of styles.

Keywords: non-photorealistic rendering, line art, multitexturing, chicken-and-egg problem

1 Introduction

In drawings, hatching can simultaneously convey lighting, suggest material properties, and reveal shape. Hatching generally refers to groups of strokes with spatially-coherent direction and quality. The local density of strokes controls tone for shading. Their character and aggregate arrangement suggests surface texture. Lastly, their direction on a surface often follows principal curvatures or other natural parametrization, thereby revealing bulk or form. This paper presents a method for real-time hatching suitable for interactive non-photorealistic rendering (NPR) applications such as games, interactive technical illustrations, and artistic virtual environments.

In interactive applications, the camera, lighting, and scene change in response to guidance by the user, requiring dynamic rearrangement of strokes in the image. If strokes are placed independently for each image in a sequence, lack of temporal coherence will give the resulting animation a flickery, random look in which individual strokes are difficult to see. In order to achieve temporal coherence among strokes, we make a choice between two obvious frames of reference: image-space and object-space. Image-space coherence makes it easier to maintain the relatively constant stroke width that one expects of a drawing. However, the animated sequence can suffer from the "shower-door effect" – the illusion that the user is viewing the scene through a sheet of semi-transmissive glass in which the strokes are embedded. Furthermore, with image-space coherence, it may be difficult to align stroke directions with the surface parametrization in order to reveal shape. Instead, we opt for object-space coherence. The difficulty when associating strokes with the object is that the width and density of strokes must be adjusted to maintain desired tones, even as the object moves toward or away from the camera.

URL: http://www.cs.princeton.edu/gfx/proj/hatching

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA

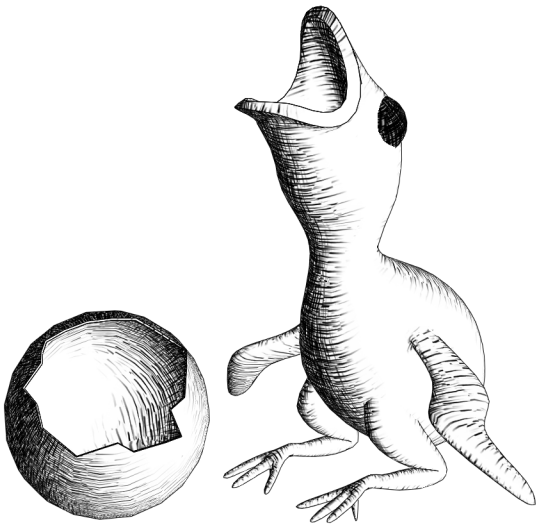


Figure 1: 3D model shaded with hatching strokes at interactive rate.

In short, hatching in an interactive setting presents three main challenges: (1) limited run-time computation, (2) frame-to-frame coherence among strokes, and (3) control of stroke size and density under dynamic viewing conditions. We address these challenges by exploiting modern texture-mapping hardware.

Our approach is to pre-render hatch strokes into a sequence of mip-mapped images corresponding to different tones, collectively called a *tonal art map* (TAM). At runtime, surfaces in the scene are textured using these TAM images. The key problem then is to locally vary tone over the surface while maintaining both spatial and temporal coherence of the underlying strokes. Our solution to this problem has two parts. First, during offline creation of the TAM images, we establish a nesting structure among the strokes, both between tones and between mip-map levels. We achieve tone coherence by requiring that strokes in lighter images be subsets of those in darker ones. Likewise, we achieve resolution coherence by making strokes at coarser mip-map levels be subsets of those at finer levels. Second, to render each triangle face at runtime we blend several TAM images using hardware multitexturing, weighting each texture image according to lighting computed at the vertices. Since the blended TAM images share many strokes, the effect is that, as an area of the surface moves closer or becomes darker, existing strokes persist while a few new strokes fade in.

As observed by Girshick et al. [3], the *direction* of strokes drawn on an object can provide a strong visual cue of its shape. For surfaces carrying a natural underlying parametrization, TAMs may be applied in the obvious way: by letting the rendered strokes follow isoparameter curves. However, to generate hatching over a surface of arbitrary topology, we construct for the given model a *lapped texture* parametrization [17], and render TAMs over the resulting set of parametrized patches. The lapped texture construction aligns the patches with a direction field on the surface. This direction field may be either guided by the user to follow semantic features of the shape, or aligned automatically with principal curvature directions using a method similar to that of Hertzmann and Zorin [7].

The specific contributions of this paper are:

- the introduction of tonal art maps to leverage current texturing hardware for rendering strokes (§3),
- an automatic stroke-placement algorithm for creating TAMs with stroke coherence at different scales and tones (§4),
- a multitexturing algorithm for efficiently rendering TAMs with both spatial and temporal coherence (§5), and
- the integration of TAMs, lapped textures and curvature-based direction fields into a real-time hatching system for shading 3D models of arbitrary topology (§6).

2 Previous work

Much of the work in non-photorealistic rendering has focused on tools to help artists (or non-artists) generate imagery in various traditional **media or styles** such as impressionism [5, 12, 14], technical illustration [4, 19, 20], pen-and-ink [1, 21, 22, 26, 27], and engraving [16]. Our work is applicable over a range of styles in which individual strokes are visible.

One way to categorize NPR methods would be to consider the **form of input** used. One branch of stroke-based NPR work uses a reference *image* for color, tone, or orientation, e.g. [5, 21, 22]. For painterly processing of *video*, Litwinowitcz [12] and later Hertzmann and Perlin [6] addressed the challenge of frame-to-frame coherence by applying optical flow methods to approximate object-space coherence for strokes. Our work follows a branch of research that relies on *3D models* for input.

Much of the work in **3D** has focused on creating *still images* of 3D scenes [1, 2, 20, 24, 25, 26, 27]. Several systems have also addressed off-line *animation* [1, 14], wherein object-space stroke coherence is considerably easier to address than it is for processing of video. A number of researchers have introduced schemes for *interactive* non-photorealistic rendering of 3D scenes, including technical illustration [4], "graftals" (geometric textures especially effective for abstract fur and leaves) [8, 10], and real-time silhouettes [2, 4, 7, 13, 15, 18]. While not the focus of our research, we extract and render silhouettes because they often play a crucial role in drawings.

In this paper, we introduce **tonal art maps**, which build on two previous technologies: "prioritized stroke textures" and "art maps." Described by both Salisbury et al. [21] and Winkenbach et al. [26], a *prioritized stroke texture* is a collection of strokes that simultaneously conveys material and continuously-variable tone. Prioritized stroke textures were designed for creating still imagery and therefore do not address either efficient rendering or temporal coherence. One might think of TAMs as an organization for sampling and storage of prioritized stroke textures at a discrete set of scales and tones, for subsequent real-time, temporally-coherent rendering. TAMs also build on the "art maps" of Klein et al. [9], which adjust screen-space stroke density in real time by using texture-mapping hardware to slowly fade strokes in or out. TAMs differ from art maps in two ways. First, art maps contain varying tone within each image, whereas TAMs are organized as a *series* of art maps, each establishing a single tone. Second, TAMs provide stroke coherence across both scales and tones in order to improve temporal coherence in the resulting imagery.

Three previous systems have addressed **real-time hatching** in 3D scenes. Markosian et al. [13] introduced a simple hatching style indicative of a light source near the camera, by scattering a few strokes on the surface near (and parallel to) silhouettes. Elber [2] showed how to render line art for parametric surfaces in real time; he circumvented the coherence challenge by choosing a fixed density of strokes on the surface regardless of viewing distance. Finally, Lake et al. [11] described an interactive hatching system with stroke coherence in image space (rather than object space).

3 Tonal Art Maps

Drawing hatching strokes as individual primitives is expensive, even on modern graphics hardware. In photorealistic rendering, the traditional approach for rendering complex detail is to capture it in the form of a texture map. We apply this same approach to the rendering of hatching strokes for NPR. Strokes are pre-rendered into a set of texture images, and at runtime the surface is rendered by appropriately blending these textures.

Whereas conventional texture maps serve to capture material detail, hatching strokes in hand-drawn illustrations convey both material and *shading*. We therefore discretize the range of tone values, and construct a sequence of hatch images representing these discrete tones. To render a surface, we compute its desired tone value (using lighting computations at the vertices), and render the surface unlit using textures of the appropriate tones. By selecting multiple textures on each face and blending them together, we can achieve fine local control over surface tone while at the same time maintaining spatial and temporal coherence of the stroke detail. We present the complete rendering algorithm in Section 5.

Another challenge when using conventional texture maps to represent non-photorealistic detail is that scaling does not achieve the desired effect. For instance, one expects the strokes to have roughly uniform screen-space width when representing both near and far objects. Thus, when magnifying an object, we would like to see more strokes appear (so as to maintain constant tone over the enlarged screen-space area of the object), whereas ordinary texture mapping simply makes existing strokes larger. The *art maps* of Klein et al. [9] address this problem by defining custom mip-map images for use by the texturing hardware. In this paper, we use a similar approach for handling stroke width. We design the mip-map levels such that strokes have the same (pixel) width in all levels. Finer levels maintain constant tone by adding new strokes to fill the enlarged gaps between strokes inherited from coarser levels. We perform this mip-map construction for the stroke image associated with each tone. We call the resulting sequence of mip-mapped images a *tonal art map* (TAM).

A tonal art map consists of a 2D grid of images, as illustrated in Figure 2. Let (ℓ, t) refer to indices in the grid, where the row index ℓ is the mipmap level ($\ell = 0$ is coarsest) and the column index t denotes the tone value ($t = 0$ is white).

Since we render the surface by blending between textures corresponding to different tones and different resolutions (mipmap levels), it is important that the textures have a high degree of coherence. The art maps constructed in [9] suffered from lack of coherence, because each mipmap level was constructed *independently* using "off-the-shelf" NPR image-processing tools. The lack of coherence between the strokes at the different levels create the impression of "swimming strokes" when approaching or receding from the surface.

Our solution is to impose a stroke *nesting property*: all strokes in a texture image (ℓ, t) appear in the same place in all the darker images of the same resolution and in all the finer images of the same tone – i.e. every texture image (ℓ', t') where $\ell' \geq \ell$ and $t' \geq t$ (through transitive closure). As an example, the strokes in the gray box in Figure 2 are added to the image on its left to create the image on its right. Consequently, when blending between neighboring images in the grid of textures, only a few pixels differ, leading to minimal blending artifacts (some gray strokes).

Figure 3 demonstrates rendering using tonal art maps. The unlit sphere in Figure 3a shows how the mip-maps gradually fade out the strokes where spatial extent is compressed at the parametric pole. The lit cylinder in Figure 3b shows how TAMs are able to capture smoothly varying tone. Finally, Figure 3c shows the combination of both effects. The texture is applied to these models using the natural parametrization of the shape and rendered using the blending scheme described in Section 5.

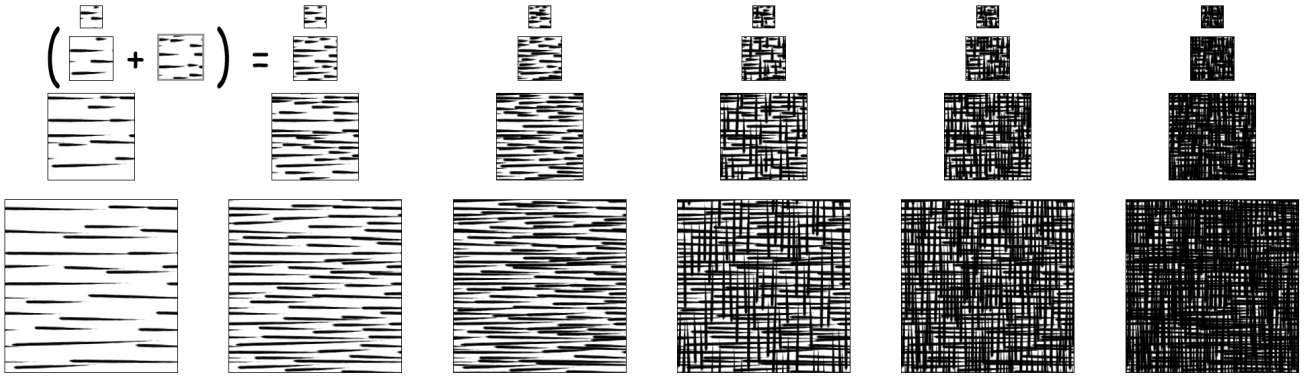


Figure 2: A Tonal Art Map. Strokes in one image appear in all the images to the right and down from it.

4 Automatic generation of line-art TAMs

The concept of tonal art maps is quite general and can be used to represent a variety of aesthetics (e.g. pencil, crayon, stippling, and charcoal). The grid of images in the TAM can be either hand-drawn or generated automatically. In this section we present our technique for automatically generating a line-art TAM.

Recall that the strokes in the resulting TAM should satisfy the nesting property described in the previous section. Our approach is to fill the TAM images in top-to-bottom, left-to-right order. For each column, we begin by copying all the images from the column to its left (or setting all images to white when starting at the leftmost column). This ensures that all strokes in images with lighter tone also appear in images with darker tone. Within the column, we then consider each image in top-to-bottom (coarse-to-fine) order. We repeatedly add strokes to the image until the mean image tone reaches the tone value associated with that column. Each stroke drawn in the image is also added to all the lower (finer) images in the current column. This ensures that strokes in coarser images always appear in finer images.

The naïve approach of random stroke selection leads to a non-uniform distribution of strokes, as illustrated in the figure to the right. Typically an artist would space the strokes more evenly. To get even spacing, we generate multiple randomly-placed candidate strokes and select the "best-fitting" one. In the case of the TAM shown in Figure 2, the number of candidates is set to 1000 for the light-tone images where spacing is important, and is gradually reduced to 100 for the dark-tone images where strokes overlap. The length of each candidate stroke is randomly set to between 0.3 and 1.0 times the width of the image, and its orientation has a tiny random perturbation from the mean. Other TAMs shown in Figure 5 use different parameters.

Our measure of "best-fitting" includes both progress towards the desired tone value and hatching uniformity. For each candidate stroke s_i we measure its progress as follows. For the hatch image at each level ℓ of the TAM that would receive s_i, we find the average tones, T_ℓ and T^i_ℓ, respectively, of the hatch image drawn with all previously chosen strokes and with s_i added. The sum $\sum_\ell (T^i_\ell - T_\ell)$ expresses the darkness that this stroke would add to all the hatch images in this column – essentially a measure of the progress toward the desired tone at all levels of the TAM.

To achieve greater hatching uniformity, we maintain an image pyramid for each unfilled image in the TAM column, and find the cumulative effect of adding s_i at all levels p of these pyramids. Even though s_i might lie next to, but not overlap, an existing stroke at fine levels of a pyramid, it will overlap at coarser levels. Thus we find the average tones $T_{p\ell}$ and $T^i_{p\ell}$ (without and with s_i, respectively) over each level p of every pyramid, and sum $\sum_{p,\ell} (T^i_{p\ell} - T_{p\ell})$.

Finally, this simultaneous measure of progress and hatching uniformity favors longer strokes over shorter strokes, even at the expense of overlapping a previous stroke. Therefore, we normalize the "goodness" of the stroke by its length, measuring the goodness of s_i as: $\frac{1}{||s_i||} \sum_{p,\ell} (T^i_{p\ell} - T_{p\ell})$. With optimized stroke placement, the hatch images are much more uniform, as illustrated in Figures 2.

Artists often use parallel strokes to represent lighter tones, and switch to cross-hatching for darker tones. Our TAM construction process supports cross-hatching, as demonstrated in Figure 2. For some number of leftmost columns (e.g. 3), we add only horizontal strokes. Then, beginning with some middle column (e.g. 4^{th}), we add only vertical strokes for cross-hatching. Finally, some number of columns later (e.g. 6^{th}), we add *both* vertical and horizontal strokes to best fill the remaining gaps.

In our current implementation, stylistic variation (Figure 5) between different TAMs comes from the variation in angle, cross-hatching schedule, range of lengths, and choice of stroke (given as a grayscale image). For now we only use grayscale strokes, to enable an efficient rendering scheme described in the next section. The finest hatch images are 256×256, and are toroidal to allow for tiled rendering over parametrized surfaces.

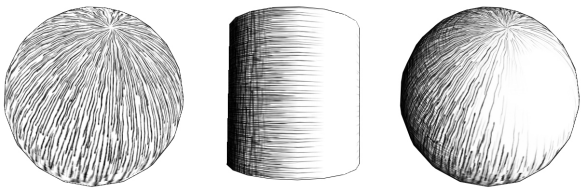
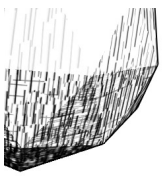


Figure 3: Rendering using tonal art maps. (a) illustrates resolution changes using custom mipmaps; (b) shows smooth changes in tone and (c) shows their interplay on a shaded sphere.

5 Rendering with TAMs

The nesting structure in the TAM image grid helps to provide temporal coherence during animation. The mip-mapping feature of hardware texturing automatically blends between different resolution levels of the TAM. What is missing, however, is a smooth transition in the tonal dimension of the TAM. We achieve smooth transitions using a 6-way blending scheme that we will describe shortly. However, we first present three intermediate approximations.

First, if for each face we select a single tone column from the TAM (equivalent to flat shading with a quantized palette), our rendering suffers from both spatial discontinuities across edges (see adjacent figure) and temporal discontinuities ("pops" in tone). Second, the analog of conventional flat shading is to interpolate between two



consecutive tone columns using a single blend ratio for all pixels in a face. This effect is temporally smooth, but looks faceted. For spatial coherence we need to blend vertex contributions across each face. Thus, our third scheme selects a single tone column per vertex and blends tones across faces. This scheme is spatially smooth but suffers from temporal discontinuities due to tonal value rounding. (This third scheme is equivalent to Gouraud-shading with severely quantized vertex lighting computations.)

To obtain both spatial and temporal coherence, we bracket the continuous tone at each mesh vertex (blending between two consecutive tone images). Then, for each mesh face, we gather the blended textures at the three vertices and spatially combine them across the face (using a barycentric sum as in Gouraud shading). Since both the 2-way tonal blend and 3-way spatial blend are linear combinations, this amounts to linearly blending 6 images across the face (or 12 images when taking into account mip-mapping). The process is illustrated in Figure 4. While this complex blend operation may initially appear daunting, there are several ways of implementing it efficiently on commodity graphics hardware. We next present two such schemes.

Single-pass 6-way blend. The 6-way blend can be implemented in a single rendering pass. Several existing graphics card support 2 texture reads during rasterization (and that number will soon jump to 4). Exploiting the fact that our TAM images are grayscale, we pack 6 consecutive tone images in the *R*,*G*,*B* channels of two texture images. The diffuse and specular fragment colors encode the blend ratios. We set these color values at vertices, and the hardware interpolates them at pixel locations. The 6 tones are combined by adding two dot products between the texture colors and the fragment colors. The first multitexture stage performs the two dot products, while the second stage adds the results.

Hertzmann and Zorin [7] observe that effective drawings can use a limited palette of tones, and their scheme targets 4 tone levels. For our examples, we chose to define 6 tone levels, so that the TAM images can be packed in two textures used over all triangles. Since the texture state remains constant, we use triangle strips for efficiency. We store vertex geometry and texture coordinates on the graphics card in vertex arrays, whereas the diffuse and specular vertex colors must be copied from main memory for every frame. With newly-emerging graphics cards, however, these could also be computed on the GPU using vertex shaders.

Our set of 6 tones do not include white – the "paper" color. However, our interpolation method provides for an implicit white as follows. Since the coefficients of a convex combination sum to 1.0, by specifying 6 independent coefficients we can implicitly blend 7 tones – the 7^{th} tone being 0.0 (black). Since we prefer white, we negate the texture inputs before the dot products, as well as the resulting sum. Thus, the images in our paper are rendered using 6-column TAMs (containing strokes) as well as implicit white.

When hatching surfaces of arbitrary topology using lapped textures, rendering is complicated by the need to modulate each patch texture by the patch outline alpha mask. We address this problem in Section 6.

Triple-pass 2-way blend. If the TAM images contain colored strokes, or if the artist desires more than 6 TAM tone images, an alternative scheme is to accumulate texture over each face in 3 passes. The scheme begins by drawing the whole mesh in black. Next it renders each triangle 3 times, adding the contribution of each vertex to the framebuffer. For each vertex we interpolate between the two TAM columns that bracket the vertex tone. A simple optimization is to organize the faces in triangle fans around each vertex.

Tone thresholding. The 6-way blend scheme obtains coherence, but results in the appearance of some gray-toned strokes. Such strokes look natural for a variety of artistic styles including pencil and charcoal, but are not ideal for some ink-based strokes. To more closely match ink strokes, we have experimented with thresholding the tone values written to the framebuffer. For the single-pass blending scheme, we can configure the multitexturing register combiners to obtain the transfer function **clamp**$(8t - 3.5)$, which closely models a step function. This transfer function succeeds in making new strokes appear as gradually lengthening black strokes instead of gradually darkening gray strokes. Unfortunately, as shown on the accompanying video, it has the side-effect of introducing jaggies along the carefully antialiased edges of the visible strokes. It may be possible to alleviate these artifacts through framebuffer supersampling.

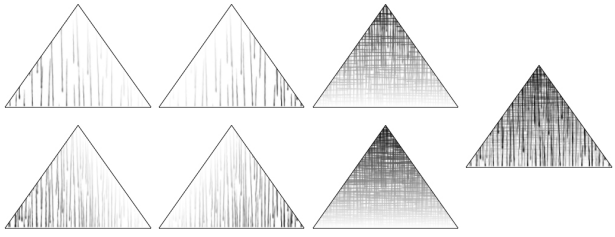


Figure 4: Illustration of the blending of 6 TAM images on a triangle face. Each column (a–c) corresponds to a vertex and shows the images for the floor and ceiling of the vertex tone. The resulting sum is shown in (d).

6 Applying TAMs to arbitrary surfaces

For parametrized surfaces, one can apply the TAM rendering algorithm described in Section 5 directly. For example, the sphere and cylinder shown in Figure 3 were textured using a simple tiling of a toroidal TAM. However, for surfaces that lack natural global parametrizations, or for which this parametrization is too distorted, we use lapped textures.

To review, lapped textures were introduced by Praun et al. [17] as a way to texture surfaces of arbitrary topology without the need for a continuous global parametrization. The key observation is that the surface can be covered by a collection of overlapping patches. All the patches are constructed as topological discs, and are parameterized over the plane with little distortion. For many textures, including our TAMs, the boundaries between the patches are not noticeable, being masked by the high frequency detail of the textures. Visual masking is further helped by constructing the patches with irregularly shaped boundaries and with small alpha ramps that "feather" the boundaries.

Hatch images are ideal for use with lapped textures, because they lack low-frequency components (which otherwise might reveal the patch boundaries). To render a surface as a lapped texture using TAMs, we apply the algorithm is Section 5 for each of the patches, shown in random tone to the right. We modify the original lapped texture rendering algorithm to adapt it for hatch rendering by decoupling the patch outline from the stroke textures, storing them separately. When rendering a patch, we first render its triangles textured using the patch outline mask, in order to store this mask into the alpha channel of the framebuffer. Next, when rendering the patch triangles using the TAM texture images, the blend operation is set to add the pixels being rendered, multiplied by the alpha value already present in the framebuffer. (During this operation, only the color channels of the framebuffer are written.) As an alternative

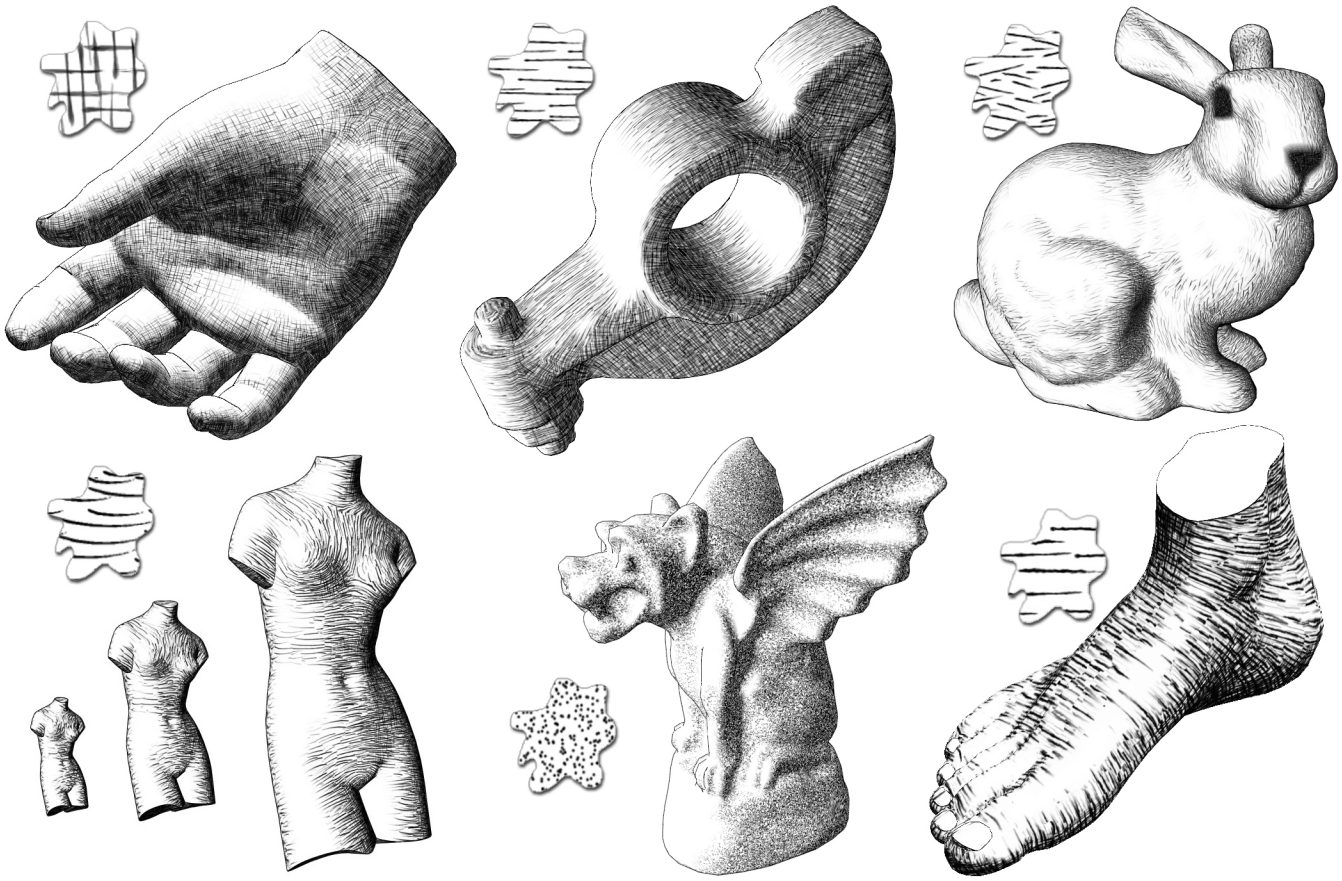


Figure 5: Results. Six models rendered with different TAMs, indicated in the inset texture patches.

useful for rendering into visuals that do not have an alpha channel, the modulation by the patch outline may be done using a texture unit. To implement this on graphics cards that allow only two texture reads per pixel, the consecutive tonal levels to be averaged have to be packed inside a single texture (see Section 5).

Decoupling the patch shape from the hatch textures presents several advantages. First, it reduces memory consumption, since we only have to store the outline once, and not replicate it for every image in the TAM. Second, having a separate texture transform for the hatch texture allows us to rotate and scale the strokes within a patch at runtime, without the need to recompute the lapped parametrization. Third, the stroke textures can be tiled in order to cover large patches. Finally, as a consequence of having a different scale for the outline than for the strokes, we can place different size patches on differents parts of the surface (smaller patcher in areas with high curvature, larger patches in flat regions), achieving better coverage while using just a single patch outline. In the original lapped textures work, having different size patches could only be achieved using multiple outlines.

Anisotropic lapped textures need a direction field for the parametrization. In our project, the field governs the direction of strokes on the surface. Girshick et al. [3] observed that stroke direction augments the viewer's perception of shape, and that principal curvatures are a natural choice for direction. Hertzmann and Zorin [7] point out that in many drawings the hatch direction follows the curvature of the overall shape, whereas fine details are expressed through tonal variation. Consequently, we compute curvatures on the faces of a simplified version of the mesh, smooth the resulting direction field, and then resample it on the original mesh. For every face of the coarse mesh, we gather the set of vertices adjacent to any of the face's vertices. We then fit a quadric through these vertices, and finally compute principal curvature directions for this quadric. The error residual from the fit, as well as the ratio of the curvatures, provides a measure of confidence for the computed directions. Now that we have defined these directions for every face, we smooth the field, blending directions from high-confidence areas into low-confidence areas. We use a global non-linear optimization similar to the one described by Hertzmann and Zorin [7], but adapted for 180° rather than 90° symmetries.

7 Results and Discussion

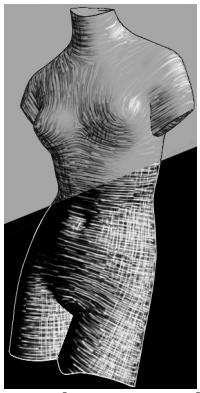


Figure 5 shows six models rendered by our system, each using using a different style TAM (shown inset). As shown to the right, by simply changing the tone values in the columns of the TAM and adjusting a tone transfer function, the renderer can also be used to produce black-and-white crayon on gray paper (above), or to simulate a white-on-black "scratchboard" style (below). The original models range from 7,500 to 15,000 faces, while overlapping patches due to lapped textures increase the range by roughly 50%. Our prototype renders these models at 20 to 40 frames per second on a 933MHz Pentium III with OpenGL/GeForce2 graphics. To use the efficient rendering scheme described in Section 5 we have restricted our TAMS to grayscale; however, the emergence of new graphics cards with more multitexture stages will enable rendering of color TAMs with comparable frame rates.

The direction field on the "rocker arm" model (Figure 5, upper-middle) was specified by hand in order to follow semantic features of the shape, whereas the rest of the models used direction fields computed with the method described in Section 6.

As demonstrated on the accompanying video tape, the strokes exhibit temporal coherence when we animate the models and light. Furthermore, the hatching density is roughly uniform across each model, regardless of the scale of the model in the image plane (for example, see the venus in Figure 5).

While it is not the focus of our research, we identify silhouette edges using the method of Sander et al. [23] and draw them as simple polylines.

8 Summary and future work

We introduced the tonal art map representation, and showed that it permits real-time rendering of stroke-based textures. We described a texture blending scheme for maintaining spatial and temporal coherence during changes in view and lighting. Although it involves interpolating 6 textures on each triangle, we have demonstrated that the blending can be implemented efficiently on existing multitexturing hardware. We focused our results on line-art style renderings, and presented an algorithm for automatically constructing tonal art maps in that style. When combined with a lapped texture parametrization, our framework permits real-time NPR rendering for many complex shapes. There remain a number of interesting areas for future work:

Stroke growth. Currently, the tone in the tonal art map is adjusted solely through the number of strokes; one could also consider adjusting the length and width of the strokes.

Automatic indication. Artists commonly use *indication*, strokes drawn in selected areas that imply texture over a broader surface. Addition of this technique would enhance real-time hatching.

More general TAMs. Although this paper emphasizes line-art hatching, we believe our framework extends to a much richer set of artistic styles. In addition, other axes besides tone could be used to parametrize art maps; a particularly interesting case would be to construct lighting-dependent textures, such as bricks whose highlights adjust to relative light position (as demonstrated in [26]).

View-dependent stroke direction. Stroke direction is currently determined during the parametrization preprocess. It might be useful to vary stroke direction at runtime as the viewpoint moves. One ambitious goal would be to attempt to hide direction field singularities. It is well known that one cannot smoothly comb a hairy ball *statically*, but perhaps dynamic combing could keep the poles hidden from view at all times.

Acknowledgments

The research reported here was supported by Microsoft Research and a NSF Career Grant. We appreciate the discussion and guidance of Georges Winkenbach and Lee Markosian, as well as help from Grady Klein. The models shown are from Julie Dorsey, Viewpoint, Cyberware, and Stanford University.

References

[1] DEUSSEN, O., AND STROTHOTTE, T. Computer-generated pen-and-ink illustration of trees. *Proceedings of SIGGRAPH 2000*, 13–18.

[2] ELBER, G. Interactive line art rendering of freeform surfaces. *Computer Graphics Forum 18*, 3 (September 1999), 1–12.

[3] GIRSHICK, A., INTERRANTE, V., HAKER, S., AND LEMOINE, T. Line direction matters: An argument for the use of principal directions in 3D line drawings. *Proceedings of NPAR 2000*, 43–52.

[4] GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. Interactive technical illustration. *1999 ACM Symposium on Interactive 3D Graphics*, 31–38.

[5] HAEBERLI, P. E. Paint by numbers: Abstract image representations. *Proceedings of SIGGRAPH 90*, 207–214.

[6] HERTZMANN, A., AND PERLIN, K. Painterly rendering for video and interaction. *Proceedings of NPAR 2000*, 7–12.

[7] HERTZMANN, A., AND ZORIN, D. Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, 517–526.

[8] KAPLAN, M., GOOCH, B., AND COHEN, E. Interactive artistic rendering. *Proceedings of NPAR 2000*, 67–74.

[9] KLEIN, A., LI, W., KAZHDAN, M., CORREA, W., FINKELSTEIN, A., AND FUNKHOUSER, T. Non-photorealistic virtual environments. *Proceedings of SIGGRAPH 2000*, 527–534.

[10] KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. Art-based rendering of fur, grass, and trees. *Proceedings of SIGGRAPH 99* (August 1999), 433–438.

[11] LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. Stylized rendering techniques for scalable real-time 3d animation. *Proceedings of NPAR 2000*, 13–20.

[12] LITWINOWICZ, P. Processing images and video for an impressionist effect. *Proceedings of SIGGRAPH 97*, 407–414.

[13] MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. Real-time nonphotorealistic rendering. *Proceedings of SIGGRAPH 97*, 415–420.

[14] MEIER, B. J. Painterly rendering for animation. *Proceedings of SIGGRAPH 96*, 477–484.

[15] NORTHRUP, J. D., AND MARKOSIAN, L. Artistic silhouettes: A hybrid approach. *Proceedings of NPAR 2000*, 31–38.

[16] OSTROMOUKHOV, V. Digital facial engraving. *Proceedings of SIGGRAPH 99*, 417–424.

[17] PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. Lapped textures. *Proceedings of SIGGRAPH 2000*, 465–470.

[18] RASKAR, R., AND COHEN, M. Image precision silhouette edges. *1999 ACM Symposium on Interactive 3D Graphics*, 135–140.

[19] ROSSL, C., AND KOBBELT, L. Line-art rendering of 3D models. *Proceedings of Pacific Graphics 2000*.

[20] SAITO, T., AND TAKAHASHI, T. Comprehensible rendering of 3D shapes. *Proceedings of SIGGRAPH 90*, 197–206.

[21] SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. Interactive pen-and-ink illustration. *Proceedings of SIGGRAPH 94*, 101–108.

[22] SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. Orientable textures for image-based pen-and-ink illustration. *Proceedings of SIGGRAPH 97*, 401–406.

[23] SANDER, P., GU, X., GORTLER, S., HOPPE, H., AND SNYDER, J. Silhouette clipping. *Proceedings of SIGGRAPH 2000*, 335–342.

[24] SOUSA, M. C., AND BUCHANAN, J. W. Observational model of blenders and erasers in computer-generated pencil rendering. *Proceedings of Graphics Interface '99*, 157 – 166.

[25] SOUSA, M. C., AND BUCHANAN, J. W. Computer-generated graphite pencil rendering of 3D polygonal models. *Computer Graphics Forum 18*, 3 (September 1999), 195–208.

[26] WINKENBACH, G., AND SALESIN, D. H. Computer-generated pen-and-ink illustration. *Proceedings of SIGGRAPH 94*, 91–100.

[27] WINKENBACH, G., AND SALESIN, D. H. Rendering parametric surfaces in pen and ink. *Proceedings of SIGGRAPH 96*, 469–476.