Fine Tone Control in Hardware Hatching

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

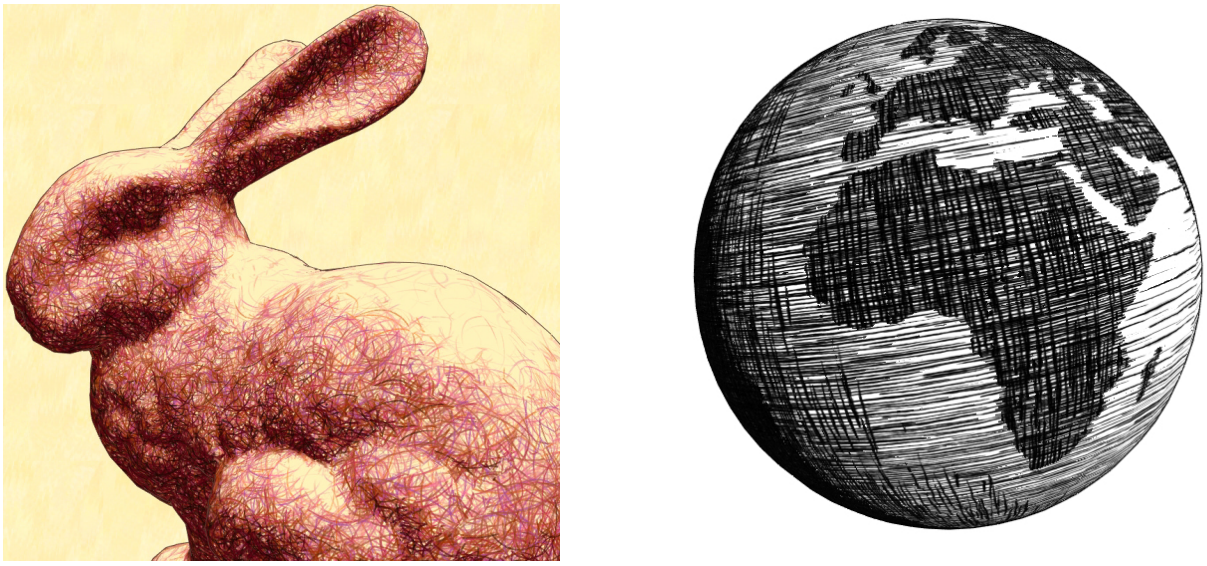


Figure 1: Bunny rendered using color volume texture; globe rendered using threshold textures (modulated per-pixel).

Abstract

Recent advances in NPR have enabled real-time rendering of 3D models shaded with hatching strokes for use in interactive applications. The key challenges in real-time hatching are to convey tone by dynamically adjusting stroke density, while controlling stroke size and maintaining frame-to-frame coherence. In this paper, we introduce two new real-time hatching schemes that leverage recent advances in texture mapping hardware. Both schemes provide enhanced control of tone, thereby avoiding blending or aliasing artifacts present in previous systems. The first scheme, which relies on volume rendering hardware, admits the use of color. The second scheme, which uses pixel shaders, allows per-pixel lighting operations such as texture modulation. Both schemes run at interactive rates on inexpensive PC graphics cards.

1. Introduction

A variety of non-photorealistic rendering styles use hatching strokes to convey tone (through stroke density), suggest material (through stroke arrangement), and reveal shape (through stroke orientation). Interactivity presents a number of challenges for applications using non-photorealistic rendering: (1) limited run-time computation, (2) frame-to-frame coherence among strokes, (3) control of stroke size and density under dynamic viewing conditions. Two recent algorithms have leveraged advances in hardware texturing capabilities to enable the use of hatching strokes in interactive applications [2][7]. However, to achieve fine tone control, these systems have suffered from a tradeoff between temporal aliasing and blending artifacts.

In this paper we present two new real-time hatching schemes that extend our previous work on *tonal art maps* (TAMs) [7]. By providing greater control over the introduction and removal of strokes in the image plane, both schemes offer finer control over tone. In addition, each new scheme exploits features of modern texture mapping hardware to enable stroke-based rendering effects that were unavailable with previous methods (Figure 1):

- The first scheme exploits *volume texturing* hardware to permit finer tone control, as well as use of **color** hatching strokes.
- The second scheme extends the *texture thresholding* method of Freudenberg [2], by using multiple thresholds to reduce aliasing artifacts while permitting **per-pixel lighting** operations.

The remainder of this paper is organized as follows. Section 2 offers a brief survey of related work. Section 3 describes in detail the implementation of the two new schemes, and offers some comparisons. Section 4 describes a new method for creating TAMs with color as well as more tonal and character variation than in our previous method. Finally, Section 5 presents results and Section 6 concludes with areas of future work.

2. Related work

There have been a number of systems for NPR using hatching.

Off-line hatching. Several systems address the problem of generating high-quality hatching for static scenes in an off-line process. Saito and Takahashi [8] describe a method for post-processing the framebuffer to overlay image-space strokes. Winkenbach and Salesin [12], and Salisbury et al. [9] introduce *prioritized stroke textures*, which map tone values to arrangements of strokes, and present impressive examples of computer-generated hatching. Sousa and Buchanan [10][11] concentrate on the technical aspects of physically simulating real media such as pencil, crayon, blenders, and erasers. Hertzmann and Zorin [3] create high-quality silhouettes, and describe an image-space stroke placement scheme for cross-hatching.

Real-time hatching. A few recent systems have addressed real-time hatching. Markosian et al. [5] introduce 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 [1] shows how to render line art for parametric surfaces in real time; he renders objects by choosing a fixed density of strokes on the surface. Lake et al. [4] describe an interactive hatching system with stroke coherence in image space (rather than in object space). Freudenberg's approach [2] consists of coding a stroke texture as a halftone pattern. To shade a pixel, the "height" of the corresponding location in the pattern is compared to the pixel's target tone, using a "soft" threshold function (a clamped linear function with high slope, instead of a step function). This approach inspired our own thresholding scheme in Section 3.2, which encodes multiple thresholds per texel for anti-aliasing.

Real-time hatching with TAMs. In previous work [7], we described how prioritized stroke textures could be rendered efficiently using texture hardware by precomputing a *tonal art map* (TAM). The images in a TAM capture hatching at various tones and scales. For visual continuity in an interactive system, we used multitexturing to blend the TAM images over each triangle. Due to hardware limitations, our system could support TAMs with only 6 different tone textures, and these textures were constrained to be grayscale. In this paper we propose two new rendering schemes that are able to utilize TAMs with finer resolution in the tone dimension, and one of the schemes naturally supports colored hatching.

3. New rendering schemes

We now present our two new rendering schemes, and compare their benefits and drawbacks.

3.1 Volume texture scheme

Recent graphics cards support volume textures, whereby a third texture coordinate r is added to the traditional (s,t) to perform lookup within a 3D texture space. Our first rendering scheme uses this third dimension r to encode tone. At load time, TAM images are simply stacked up along the tone axis of the texture volume.

On polygons with large tone variation, our previous scheme [7] would only do linear blending between the 2D textures corresponding to the extreme tone values to be represented, producing many gray strokes. The volume texture method, however, more effectively reproduces all the intermediate tones, since the 3D texture lookup can access all tone levels of a dense TAM. If the set of TAM images is sufficiently dense, the resulting rendering will give the illusion that strokes are added independently, rather than added in blended groups as in [7]. For this versatility however, we pay the price of larger texture memory consumption.

Another advantage of volume texturing is the support of color. Both our original scheme [7] and our texture threshold scheme (Section 3.2) maximize the number of reference tone images by packing them into the R,G,B,A channels of 2D textures. This packing limits the one-pass version of the schemes to grayscale strokes, requiring multi-pass implementations to render color.

Since we are using volume textures in a non-standard way, we need to take into account several aspects related to mipmapping. For our application, the ideal filtering behavior would treat the spatial dimensions separately from the tone dimension, i.e. maintain full tonal resolution even as spatial resolution decreases. Unfortunately, current hardware does not offer this behavior. (Disabling filtering altogether is not acceptable since it leads to aliasing.) There are two effects of letting the mipmapping in the tone dimension be influenced by spatial resolution: at coarse mipmap levels we lose both tone resolution and tone range.

The loss of tone resolution is not necessarily detrimental, as long as we start with enough resolution at the finest level. As the object takes up less screen space, it is harder to notice tone variation, so reducing the tone resolution is quite natural. In our examples, we used a 256×256×64 volume. The coarsest spatial level that we generate in our TAM is 32×32 (as strokes are not discernible in coarser levels), corresponding to a resolution of 8 in the tone dimension.

The loss of tone range is caused by the relationship between texture coordinates and texture samples: the first and last samples in a dimension do not correspond to coordinates 0 and 1 respectively, but to $1/2^{l+1}$ and $1 - 1/2^{l+1}$, for a mipmap level of resolution 2^l. For coordinates 0 and 1, texturing returns a 50% interpolation between the first (and respectively, last) sample and the border. Therefore, the range of tones that we can represent without border interpolation is different for each mipmap level. Using a texture with border compresses the overall range of tones available to us, and forces software rendering in our graphics driver. Instead, we perform a border interpolation correction using register combiners. This interpolation uses 100% border contribution for the texture coordinates extremes (0 and 1), rather than the default 50%.

When filling up the volume we need to follow the standard mipmapping sampling pattern. Consider the example where we want 64 levels at the finest resolution. We generate a TAM as in [7] with 128 columns. The finest level planes are assigned the 256×256 images with tones 1/128, 3/128, 5/128, … 127/128. The next resolution level are assigned the 128×128 images with tones 2/128=1/64, 3/64, 5/64, … 63/64, and so on (see Figure 2). Thus, each tone level appears in exactly one mipmap resolution level.

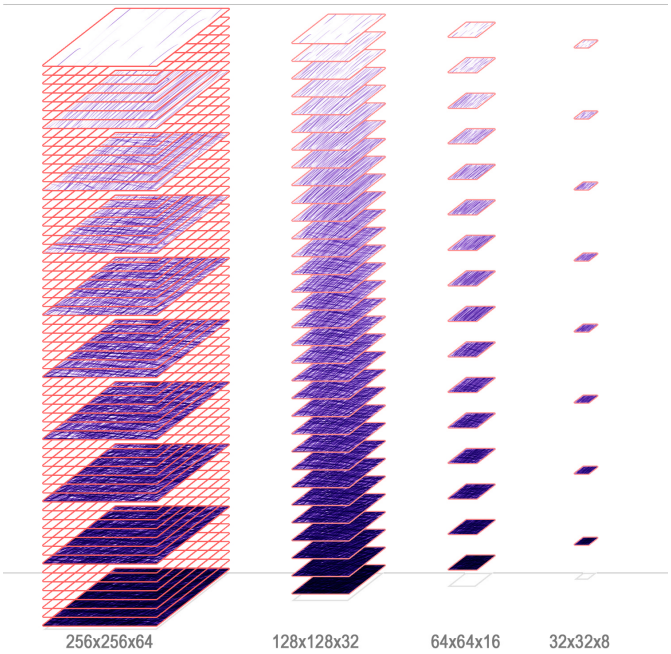


Figure 2: Tonal art map (TAM) pyramid.

3.2 Texture threshold scheme

Our second rendering scheme extends the method presented by Freudenberg [2] to produce better anti-aliasing. His method uses a halftoning approach: it stores a texture containing, for each texel, the threshold tone at which a screen pixel should change from white to black. To provide some antialiasing, the method uses a "soft" threshold function (clamp(1-4(1-(I+T))) for an input intensity I and a threshold value T). This soft threshold function works well when the change in tone is achieved by varying the width of the stroke. However, when modulating tone by adding or removing strokes, aliasing artifacts become visible, particularly with thin or overlapping strokes, and in animations. In [7], we experimented with thresholding the framebuffer to generate a traditional pen-and-ink style, but encountered similar aliasing artifacts.

When drawing correctly antialiased strokes, most of the stroke pixels should be black, but the few pixels at the stroke boundary that receive only partial coverage should be drawn in gray. (Only when a subsequent hatching stroke covers these pixels might they change to full black.) To capture this behavior, we propose to represent for each texel a piecewise constant function that maps input tones into gray values for the texel. This function therefore has several transitions, rather than a single transition as in conventional halftoning. (See Figure 3.) To render a surface, for each screen pixel we compare its tone (obtained from Gouraud interpolation) with each of the transition X values (obtained from texture lookup). We then take the sum of the heights of all the transitions that pass their comparison tests: [1]

$$\text{Pixel value} = 1 - \sum_i \begin{cases} \Delta y_i & \text{if } (1-I) > x_i \\ 0 & \text{otherwise} \end{cases}$$

This scheme introduces strokes one by one, much like the volumetric method. In fact, one can view these mapping functions as run-length encodings of rows of texels parallel to the tone dimension in the volume texture from the previous section. The volume texture is rather coherent: a texel keeps its shade for large tone intervals, between the events when different strokes touch the same texel. Since one goal of TAM generation is spatial uniformity, such events are placed as far apart as possible in the tone dimension, leading to large spans of constant values in the volume.

Since the value of I does not influence which x_i and Δy_i texture locations to address, I can in fact be a more complicated function. For instance, we can modulate I per-pixel with a texture, to produce effects such as the hatched earth globe shown in Figure 1, without affecting the triangulation of the model (as would be necessary in a scheme that could only compute I at vertices).

One problem to consider when representing such transfer functions using textures is (tri-)linear interpolation. If two neighboring texels have the same set of x_i's, interpolating the corresponding Δy_i's yields the correct result. Unfortunately, this doesn't hold for interpolating x_i values. To reduce artifacts, we try to only interpolate between close x_i values: we divide the intensity interval into several bins (not necessarily of equal length), and for each texel only allow a single transition in each bin. Consequently, when different x_i's from adjacent texels are blended together, they can differ only by at most the bin width. In our implementation we used 7 bins corresponding to at most 7 transitions, which we packed in the RGBA channels of 4 textures (we reserve 4×4 - 7×2 = 2 channels for the modulate mask and the splotch mask for a lapped parameterization [6]).

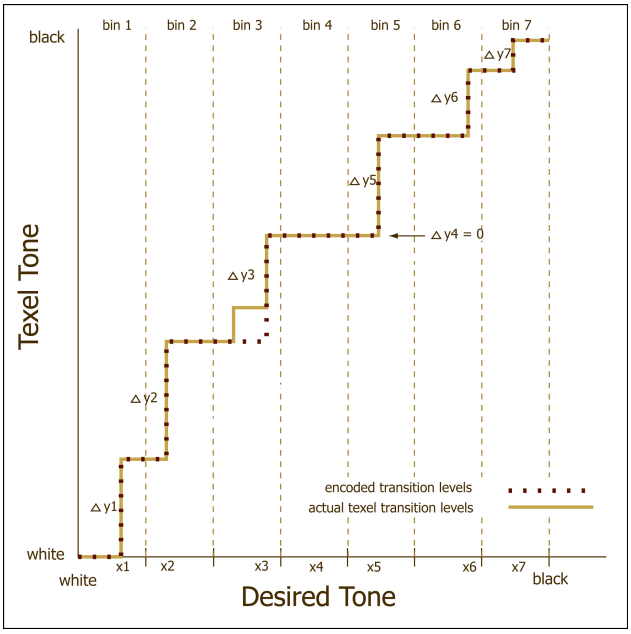


Figure 3: Transition diagram for a single texel.

Since strokes are placed uniformly across the TAM textures, it is infrequent that a texel undergoes more than one transition in the same bin. When that happens, we store Δy to be the sum of the transition heights, and randomly pick x_i from among those in the bin. Since more strokes are placed at the dark end of a TAM, we make the bins smaller at the dark end of the spectrum than at the light end. While this binning scheme tends to work well under the assumptions stated, it can lead to banding artifacts when these don't hold. When the TAM is made of many small, thin strokes, the average number of strokes touching a texel increases, and therefore, the number of transitions increases as well. In the limit, when trying to represent a continuous mapping function (no strokes — just 256 gray levels), forcing a representation using only 7 discrete transitions, spaced according to our bin distribution, produces models shaded with only 7 levels of gray, appearing in 7 bands. Choosing x_i's at random within each texel's bins helps make the band boundaries more rough than choosing the mean or average.

3.3 Comparison of the two methods

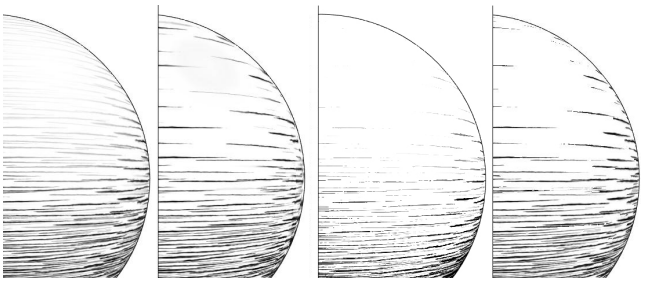
Both methods presented in this paper offer an improvement over our previous scheme [7], allowing fine control over tone representation. Since we have many more TAM columns (samples in the tone dimension), and since each pixel, rather than each vertex, determines the samples to be blended, we can give the illusion of adding (or growing) each stroke individually, rather than fading in large waves of strokes. Figure 4 shows a

[1] This function can be implemented on a GeForce3 using register combiners. Simultaneous thresholding of several values can be done by multiplying the 8-bit fixed point colors with 256 (by chaining scale_by_four()'s); while sum of products can be implemented efficiently using dot products. The double inversion (1- intensity; 1-sum) is needed because frame buffers represent "amount of white" while we want to be adding "blackness" (corresponding to black strokes on white paper). Without the inversions, our highlights (light regions on the models) will appear drawn with many overlapping white strokes on a dark canvas, rather than as a white canvas with no strokes.

comparison of the three methods, using different representations of the same 255-column TAM. Our previous method has large areas of gray strokes. In the volume rendering approach there are a few gray strokes caused by tri-linear interpolation (due primarily to mipmapping rather than tone interpolation). The threshold scheme has no gray strokes, only gray pixels around strokes for anti-aliasing. This is due to the fact that interpolation happens on the thresholds, before being compared to the tone. An artifact of this is the presence of a few thin strokes that don't get anti-aliased, since their boundary pixels do not pass the test using the interpolated thresholds.

While the threshold scheme uses less memory, it is actually slightly slower to render than the volume approach, since it involves accessing more textures per pixel. However, it gives us the opportunity for interesting per-pixel effects, such as modulating tone using a different texture. With fewer threshold bins or additional texture accesses on future hardware, one could integrate more complicated effects such as bump mapping and Phong shading. Using these effects with the volume texture rendering approach may be possible in the near future, on graphics cards that allow more complicated dependent texture accesses. Another feature that is likely to be available soon is anisotropic filtering of volume textures; its absence causes the slightly blurrier regions near the silhouettes in Figure 4b.

One of the advantages of the volume rendering approach is the ease of integrating color. This opportunity raises an interesting artistic question: what can we convey with color that we cannot convey with tone alone? While we do not offer a substantial answer to this question, we have experimented with choosing a path through the color cube, parameterized by luminosity (tone). We have chosen the hues for the colors along this path from compatible color palettes.



(a) Praun'01 [7] (b) volume (c) 1 threshold (d) 7 thresholds

Figure 4: Comparison of rendering schemes.

4. Fine-level TAM generation

Both rendering schemes require the construction of a TAM that is much denser along the tone axis than in [7] (as many as 255 tones[2] instead of 6 tones). This can be constructed using the algorithm described in our previous paper. Obtaining more tone levels does not require any more pre-processing time, since the same number of candidate strokes are still added; we simply "snapshot" more TAM images during the process.

In the remainder of this section, we present an alternative method for TAM generation that allows the user more control and more expressive power. We generate the finest levels of the TAM using a high-quality drawing package, by placing strokes in an image to achieve gradually darker tones. An automated process then replays the sequence of strokes, and selects images corresponding to the tones we want to represent at the finest TAM level. From this, we then construct the coarser levels of the TAM. This scheme works particularly well as it leverages the strengths of the artist and computer to compensate for the other's weakness. The artist need not be overly concerned with the mechanics of TAM generation; he or she simply works on a single texture, drawing a sequence of strokes until satisfied with the range of tones. The computer then handles the task of selecting subsets of strokes to form each image in the volumetric TAM (a task that is prohibitively tedious for a user to undertake).

In order to maintain coherence and tone at each level of the mipmap volume, it is important to select correlated sets of strokes. The image at a given level and tone *(l, t)* should consist of the set of strokes in the next lightest tone at the same resolution *(l, t-1)* plus some subset of the strokes used in the same tone at the next highest resolution *(l+1, t)*. Since the resolution decreased for the new level, the strokes are relatively larger, so fewer of them will be needed for the same coverage, or tone difference. For grayscale TAMs, we can simply select a prefix of the stroke sequence. However, for color TAMs representing a path parameterized by tone through the color space, taking the prefix that produces the desired tone difference will very likely give us the wrong hue. In this case, we first decimate the stroke sequence (throw out a constant fraction of randomly selected strokes), and then take the prefix. In theory, one could do a binary search to find the right fraction for each TAM or even for each image (this fraction depends on stroke properties such as aspect ratio), in practice though we have found that choosing a constant fraction works well, given that the tone steps we are trying to sample are small.

5. Results

Figure 5 shows several stills produced with our system. The accompanying video shows short animations of these models.

The hand image is drawn using a style reminiscent of chalk and charcoal. Following artistic conventions, the highlight strokes are hatched in a single direction whereas the shading also employs crosshatched strokes. The fruit bowl image uses an ink texture in which overlapping strokes combine to increase darkness. This differs from the hand image in which overlapping strokes do not darken the surface. The color stipple pattern used on the gargoyle model was an interesting artistic experiment, since it produced the widest range of reactions from the people we have shown it to. It lessens the illusion of a growing front of strokes, since the length of the stroke is short enough that new strokes are distinct from existing strokes. Finally, for the rocker arm, we tried to achieve a look evocative of mechanical sketch.

The bottom row of Figure 5 shows two examples of objects rendered using threshold textures. When these objects rotate, the strokes give the appearance of growing into the highlight regions. The crisp black and white aspect of the strokes is reminiscent of a hand drawn pen-and-ink style.

The Earth image in Figure 1 shows the integration of threshold textures and per-pixel modulation with a map texture. The bunny of Figure 1 is drawn using short arcs with random orientations. When animated, these strokes provide a different impression from the other models: since they grow in different directions, there is no illusion of an advancing front of strokes.

[2] The number 255 is due to the precision currently available in commodity graphics hardware.

All these models render at around 30-40 frames per second on our GeForce3 card. This includes time spent extracting the silhouettes and drawing the background. The original models have between 7,500 and 15,000 faces. For all models except the Earth globe and the fruit bowl, we created a lapped texture parametrization. The objects in the fruit bowl were created using spline patches, and we used their intrinsic *u,v* parametrization.

The 6-column TAM used in [7] required 800KB of texture memory. By comparison, the volume texture requires 15MB (or 20MB when keeping alpha for border correction), and the threshold textures take up 1.8MB.

6. Conclusions and future work

We have presented two methods to improve the quality of interactive hatch renderings. Both methods provide fine tone control. Volumetric textures allow for greater user expression by adding the ability to render color hatchings. Threshold textures store a discrete set of tone transitions per texel, supporting hatch rendering with fine tone control and anisotropic filtering with far less memory consumption. This is at the cost of restricting the hatched models to grayscale images.

We have found that harsh polygonal silhouettes are often the largest factor in associating the rendering with a 3D model. We would like to investigate methods draw smooth stroke-based silhouettes that complement the volumes textures.

The current implementation of volume TAMs utilizes a large amount of texture memory. Since volume TAMs have an extremely high degree of coherence by definition, it may be possible to greatly reduce the amount of memory consumption.

We would also like to investigate methods that provide a provable error bound on tone and hue among the different mipmap levels.

Furthermore, we are interested in rendering entire scenes instead of single objects. This introduces new opportunities to explore other artistic techniques as haloing and shadowing.

References

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

[2] FREUDENBERG, B. Real-Time Stroke Textures. (Technical Sketch) *SIGGRAPH 2001 Conference Abstracts and Applications*, p. 252.

[3] HERTZMANN, A., AND ZORIN, D. Illustrating Smooth Surfaces. *Proceedings of SIGGRAPH 2000*, Computer Graphics, Annual Conference Series, pp. 517–526.

[4] LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. Stylized rendering techniques for scalable real-time 3d animation. *Proceedings of NPAR2000*, pp.13–20.

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

[6] PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. Lapped Textures. *Proceedings of SIGGRAPH 2000*, Computer Graphics, Annual Conference Series, pp. 465-470.

[7] PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. REAL-TIME HATCHING. *Proceedings of SIGGRAPH 2001*, Computer Graphics, Annual Conference Series, pp. 579-584.

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

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

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

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

[12] WINKENBACH, G., AND SALESIN, D. Computer-generated pen-and-ink illustration. *Proceedings of SIGGRAPH 94*, Computer Graphics, Annual Conference Series, pp. 91–100.

Figure 5: Results. Top two rows: volume texture rendering. Bottom row: thresholds rendering.