Dynamic 2D Patterns for Shading 3D Scenes

Simon Breslav, Karol Szerszen, Lee Markosian, Pascal Barla, Joëlle Thollot

Dynamic 2D Patterns for Shading 3D Scenes

Simon Breslav, Karol Szerszen, Lee Markosian
University of Michigan

Pascal Barla, Joëlle Thollot
INRIA Grenoble University

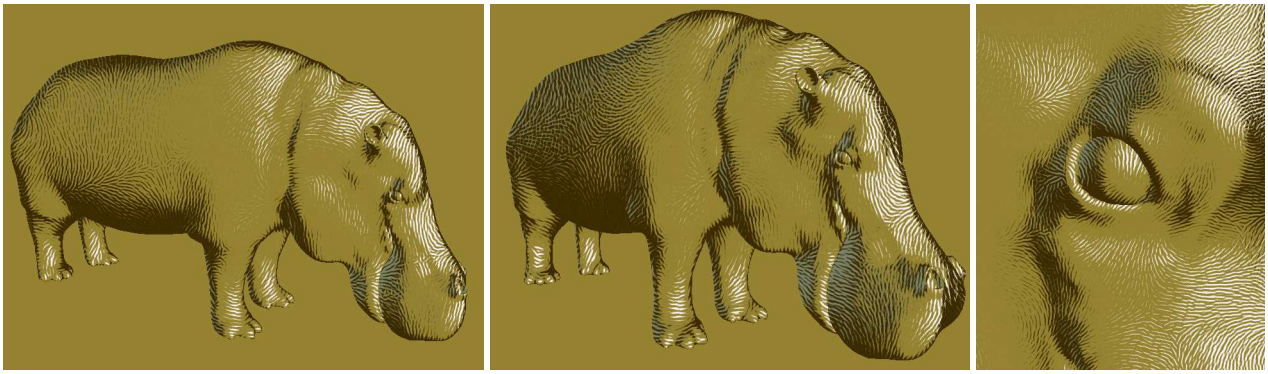

Figure 1: *A dynamic 2D pattern that scales, rotates and translates in 2D to best match the apparent motion of the underlying 3D shape.*

Abstract

We describe a new way to render 3D scenes in a variety of non-photorealistic styles, based on patterns whose structure and motion are defined in 2D. In doing so, we sacrifice the ability of patterns that wrap onto 3D surfaces to convey shape through their structure and motion. In return, we gain several advantages, chiefly that 2D patterns are more visually abstract – a quality often sought by artists, which explains their widespread use in hand-drawn images.

Extending such styles to 3D graphics presents a challenge: how should a 2D pattern move? Our solution is to transform it each frame by a 2D similarity transform that closely follows the underlying 3D shape. The resulting motion is often surprisingly effective, and has a striking cartoon quality that matches the visual style.

1 Introduction

A variety of methods have been proposed for rendering 3D scenes in non-photorealistic styles based on patterns of brush strokes or other 2D marks. These methods are typically designed to produce images that resemble traditional drawings, prints, and paintings in a range of styles. Examples include the work of Winkenbach and Salesin [1994], Meier [1996], Hertzmann and Zorin [2000], Webb *et al.* [2002], Freudenberg *et al.* [2002], and Kalnins *et al.* [2002].

In nearly all such work to date, the 2D marks are organized to follow the features of the 3D shapes that make up the scene. This helps convey a better sense of 3D shape to the viewer [Girshick et al. 2000]. Indeed, the same strategy is used as well in many hand-drawn images: strokes are often aligned with features of the shapes being depicted.

In quite a few cases, however, artists choose a different strategy – stroke directions appear independent of the underlying 3D shapes (see for example Figure 2).

Such "2D patterns" of strokes are popular because they provide greater visual abstraction. Visual abstraction takes many forms, and is of fundamental importance in non-photorealistic images because it lets the artist "stress the essential rather than the particular" [Brassard 1998]. Depending on the chosen style, better visualization of the details of a shape is not always desirable. This is especially true for less-important objects such as those in the background.

Given the prevalence of 2D patterns in hand-drawn images, it is natural to consider using them to render animated 3D scenes with computers. This presents a challenge: how should the patterns move? A trivial solution is to keep the pattern fixed in the image, but this leads to a disturbing disconnect between the movement of the pattern and the underlying object – a condition known as the "shower door" effect. On the other hand, any motion that exactly matches the image-space motion of the 3D object will result in distortions in the shape of the 2D pattern.

Our solution is to compute a shape-preserving 2D "similarity transform" combining translation, rotation, and uniform scaling to match as closely as possible the apparent motion of the surface (see Figure 1 and the accompanying video). The scene can be divided into separate objects, or patches within an object, with a separate similarity transform computed for each object or patch.

We describe a working system based on these dynamic 2D patterns to achieve a variety of styles: multi-color halftoning, hatching and stippling, and a painterly style. We address LOD transitions that prevent the 2D pattern from growing too large or small. The computation of the similarity transform is straightforward and efficient, and the system leverages programmable GPUs to render at interactive rates.

Transformed patterns can still exhibit sliding, but the method works surprisingly well for many motions, and the 2D movement has a striking cartoon quality that can match the visual style.

Figure 2: *Left to right: details from drawings by Joe Sacco, Bill Plympton and Richard Sala. Stroke patterns in many hand-drawn images like these are laid out in image space and do not follow the 3D shapes depicted by them.*

2 Related work

The "dynamic canvas" method of Cunzi *et al.* [2003] addresses a closely related problem and proposes a similar solution. The problem is how to reduce the shower door effect by transforming the background "canvas" texture used for media simulation by many 3D NPR algorithms. Their solution is to apply a 2D similarity transform that matches the image-space motion of certain 3D points in the scene.

The method has two limitations: (1) it works best in the limited case of a static scene in which all objects lie at roughly the same distance from the camera; and (2) that distance is a parameter that must be supplied by the application. In contrast, our method is automatic and handles both camera and object motion (including animated objects). It can be applied independently to each object (or part of an object) in the scene.

Bousseau *et al.* [2006] and Kaplan and Cohen [2005] also describe methods for achieving a temporally coherent dynamic canvas. These methods are automatic and account for camera motion as well as objects that move independently. Both methods track seed points on 3D surfaces and use them to generate (each frame) a canvas texture by joining small pieces of texture that follow the seed points. These methods work well with small-scale, unstructured patterns like canvas and paper textures, but cannot preserve regular or large-scale patterns like hatching and halftone screens. Eissele *et al.* [2004] propose a similar strategy for 2D halftone patterns applied to 3D scenes. Not surprisingly, the method exhibits severe temporal artifacts, with the halftone screen continually breaking apart and re-forming.

Coconu *et al.* [2006] describe a method for applying 2D hatching patterns to 3D scenes, particularly landscape models containing trees. Leaves of the trees are clustered into "high level primitives" and rendered with 2D hatching patterns that are updated each frame using a 2D similarity transform. This method is similar to the one we propose, except they compute the similarity transform by tracking a *single* 3D sample point and orientation vector each frame, whereas we use a collection of 3D sample points and a weighted least-squares optimization to compute the similarity transform that best matches the observed motion of the 3D points. We explain the details in the next section.

3 Transforming 2D patterns

We implemented several styles of "dynamic 2D patterns" in GLSL [Rost 2006]. Our halftone shader uses a previously published halftone method [Ostromoukhov 1999; Durand et al. 2001; Freudenberg et al. 2002] to generate several color layers that follow separate light sources (see for example Figure 1). We also demonstrate hatching and painterly styles in the accompanying video. Each style uses image-space texture maps to encode halftone screens, paper textures ([Kalnins et al. 2002]), or collections of hatching or paint strokes. These "2D patterns" are typically combined in several color layers to convey shading and highlights.

We transform 2D patterns as follows. In a pre-process we distribute 3D sample points over surfaces in the scene. At run time, we use the image-space positions of the samples in the current and previous frame to compute a 2D similarity transform that closely follows their observed motion. We then apply the transform to the pattern. We explain the details in the following sections.

3.1 Samples and weights

We generate sample points using the "stratified point sampling" method of Nehab and Shilane [2004]. This method achieves a relatively uniform distribution of points over each 3D surface, independent of triangulation. 3D samples are stored using barycentric coordinates relative to mesh triangles, so they remain valid when the mesh is transformed or animated.

To reduce the chance of finding too few sample points in any frame, we use a moderately dense set of samples for each object or patch (typically several hundred). Though this is far more than needed to produce a good solution, the performance cost of using additional samples is negligible, as the computation of the similarity transform is linear in the number of samples. The sampling density can be adjusted by the user, though in practice we found that the default settings work well.

Each sample is assigned a weight each frame based on approximately how much of the visible image is taken up by the bit of surface associated with the sample. To achieve this, we compute the image-space area of a disk that lies tangent to the surface at the sample location, with diameter equal to the sample spacing. We set the weight equal to this value times a "fuzzy" visibility measure of the sample, as in the "blurred depth test" of Luft and Deussen [2006], which we implemented using an "ID image" [Kalnins et al. 2002] instead of a depth-buffer. We compare this weighting scheme to a simpler one in the accomanying video and Section 4.

We handle multiple objects (or patches within an object) by considering the sample points of each object (or patch) independently. When multiple patches are used, patterns can be overlapped and blended across patch boundaries to hide the discontinuities, as explained in Section 3.6.

3.2 Least-squares solution

We use Horn's method [Horn 1987] to compute the 2D similarity transform (combining translation, rotation, and uniform scaling) that best maps the sample locations in the previous frame to the current one. Horn's paper provides in-depth derivations, but note that our 2D case is simpler than the 3D case addressed by Horn.

Below, w_i denotes the weight assigned to sample i in the current frame (see Section 3.1), but we use $w_i = 0$ when the weight assigned in the previous frame was 0. This way, we only consider samples that are visible in both the current and previous frames.

Let $\bar{\mathbf{c}}$ and $\bar{\mathbf{p}}$ denote the weighted centroids of the image-space locations of the samples in the current and previous frames, respectively (both using weights w_i). The translation is then simply $\bar{\mathbf{c}} - \bar{\mathbf{p}}$.

Let $\mathbf{c}_i$ denote the image-space location of sample i in the current frame minus $\bar{\mathbf{c}}$. Define $\mathbf{p}_i$ similarly as the location of sample i in the previous frame minus $\bar{\mathbf{p}}$.

We seek the combination of 2D rotation and uniform scaling that best maps $\mathbf{p}_i$ to $\mathbf{c}_i$, taking into account weights w_i. Note that any 2D point or vector can be viewed as a complex number – the notations $x+iy$ and (x,y) are equivalent – and that complex number multiplication achieves 2D rotation and uniform scaling.[1] We can thus formulate the problem as a weighted least squares optimization seeking the complex number $\mathbf{z}$ that minimizes the error:

$$E = \sum_i w_i |\mathbf{z}\mathbf{p}_i - \mathbf{c}_i|^2.$$

The least-squares solution is found by taking partial derivatives with respect to the two unknown components of $\mathbf{z}$, setting equal to 0, and solving. This yields:

$$\mathbf{z} = \left(\sum_i w_i \mathbf{p}_i \cdot \mathbf{c}_i, \quad \sum_i w_i \mathbf{p}_i \times \mathbf{c}_i \right) / \sum_i w_i |\mathbf{p}_i|^2$$

(Here, "$\times$" denotes the "2D cross product.")

As Horn [1987] explains, the above formulation is not symmetric, in the sense that the computed transform from $\mathbf{c}_i$ to $\mathbf{p}_i$ is not the inverse of the transform from $\mathbf{p}_i$ to $\mathbf{c}_i$ — in general, the two rotations are inverses, but the two scale factors are not. We thus adopt the "symmetric" formulation described by Horn: we multiply $\mathbf{z}$ as computed above by the following scale factor s:

$$s = |\mathbf{z}|^{-1} \left(\frac{\sum_i w_i |\mathbf{c}_i|^2}{\sum_i w_i |\mathbf{p}_i|^2} \right)^{\frac{1}{2}}$$

Without this correction, repeated zooming in and out can gradually shrink the pattern, which may be undesirable. In any case, when the cumulative scaling applied to the pattern grows too large (or small), we initiate a transition to a less-scaled version of the pattern, as explained in Section 3.4.

[1] Multiplication by a complex number $\mathbf{z}$ has a geometric interpretation: it scales by $|\mathbf{z}|$ and rotates by the angle between $(1,0)$ and $\mathbf{z}$ (hence $i^2 = -1$).

3.3 Computing texture coordinates

The texture maps used by our shaders are defined in a canonical *uv*-space. By convention, the texture fills the unit square $[0,1]x[0,1]$, and we assume it tiles seamlessly to fill all of *uv*-space. For each separately moving pattern we store an image-space point $\mathbf{o}$ corresponding to the origin in *uv*-space, and image-space vectors $\mathbf{u}$ and $\mathbf{v}$ that correspond to *uv* vectors $(1,0)$ and $(0,1)$, respectively. In other words, $\mathbf{o}$ locates the lower left corner of the pattern in the image, and $\mathbf{u}$ and $\mathbf{v}$ determine its horizontal and vertical edges.

In each frame we compute $\bar{\mathbf{p}}$, $\bar{\mathbf{c}}$ and $\mathbf{z}$ as explained in Section 3.2, then update $\mathbf{o}$, $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$\begin{aligned} \mathbf{o} &\leftarrow \bar{\mathbf{c}} + \mathbf{z}(\mathbf{o} - \bar{\mathbf{p}}) \\ \mathbf{u} &\leftarrow \mathbf{z}\mathbf{u} \\ \mathbf{v} &\leftarrow \mathbf{z}\mathbf{v} \end{aligned}$$

In the fragment shader, *uv*-coordinates are computed from the fragment location $\mathbf{q}$ using: $((\mathbf{q}-\mathbf{o})\cdot\mathbf{u}/|\mathbf{u}|^2, (\mathbf{q}-\mathbf{o})\cdot\mathbf{v}/|\mathbf{v}|^2)$. These coordinates can be used with any 2D pattern, to make it track the apparent motion of the object in image-space.

3.4 LOD transitions

When patterns are scaled very large or small they lose their original appearance. We support a kind of "level of detail" (LOD) whereby patterns are restored to nearly their original size when the cumulative scale factor falls outside of a given range. We set two scaling thresholds $1 < s_0 < s_1 < 2$. (In our examples, we used $s_0 = 1.3$ and $s_1 = 1.7$.) As cumulative scaling increases from 1 to s_0, the pattern grows larger. As scaling increases from s_0 to s_1, the pattern continues growing but fades out, while a half-size copy of the pattern fades in. We thus transition from a somewhat too-large pattern at scale factor s_0 to a somewhat too-small pattern at scale factor s_1 (since the scale factor is applied to a half-sized pattern). As the underlying scale factor continues to increase from s_1 to 2, the pattern grows back to its normal size. We combine hatching and painterly patterns by alpha blending to achieve a cross-fade; for halftone patterns we blend between halftone *screens*, causing the pattern to morph instead of fade.

3.5 Tone correction

The blended halftone screens used for LOD transitions generally don't reproduce tones exactly as the original (scaled or unscaled) screen, so to avoid tone fluctuations during LOD transitions, we use a tone correction method like the one described by Durand *et al.* [2001]. The idea is to take into account how tones will be altered by the halftone process when using a given screen, and compensate by adjusting the input tone in order to produce the desired output tone. In the case addressed by Durand *et al.*, the tone alteration function took a known analytical form that could be inverted. In our case we do not have an analytic expression for the way tones are altered by the given halftone screen, so we tabulate it in a pre-process by applying the halftone screen to each possible input tone value and measuring the result, a strategy like the one used by Salisbury *et al.* [1997] for a similar purpose.

The inverse of the tone alteration function for each halftone screen can be stored as a 1D texture (size 256*x*1) that is used in the fragment shader to remap tone values before using them in the halftone process with that screen. For LOD transitions, we compute these 1D textures at regular samples of the LOD transition parameter (we use 16 samples), resulting in a 2D texture (size 256*x*16) needed for each halftone screen.

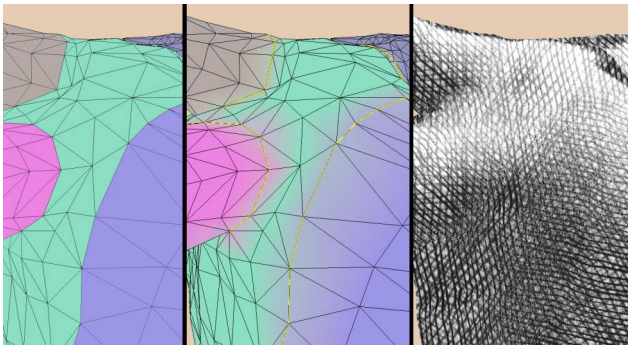

Figure 3: *Left: detail of several patches from the cow model. Middle: visualization of blending weights over patch 1-rings. Right: hatching textures blended across patch boundaries.*

3.6 Multiple patches

Our method greatly reduces the shower door effect compared to patterns that remain fixed in the image, but for some combinations of object shape and camera motions, significant sliding can still occur. To reduce sliding, the user can divide a given mesh into multiple patches, each to be rendered separately, using a similarity transform computed with samples restricted to that patch. This can reduce sliding significantly, as demonstrated in the video.

This introduces a visible discontinuity in the appearance and motion of patterns on either side of the boundary. We address that problem using a simple method for blending patterns across boundaries that effectively hides these discontinuities in many cases – especially for irregular patterns that use alpha blending, such as hatching and painterly styles. (See Figures 3 and 5.)

We blend patterns near patch boundaries using a set of blending weights – at each vertex we store one weight per patch. Weights are initially set to 1 within a patch, and to $1/k$ at boundary vertices, where k is the number of adjacent patches. To render a patch, we render all of its triangles plus those in the 1-ring outside the patch. The "strength" of the pattern is determined by the blending weight at each vertex – a low weight results in a faintly rendered pattern.

With weights initialized as described above, each pattern is drawn at half strength along its boundary, with strength falling to 0 over the outer 1-ring of the patch boundary and rising to 1 over the inner 1-ring. Where two or more patches abut, their patterns cross-fade over the 1-ring of the patch boundaries. Depending on the mesh resolution, we might want to blend patterns across wider regions. We thus let the user specify (via a slider in our GUI) the value n of the n-ring size used in patch blending. We use an iterative process to compute smoothly varying weights, as follows.

For $n-1$ iterations, we visit each vertex and, for each patch, set its weight at that vertex equal to the average of its weights over the neighboring vertices. (We compute all the new weights first, then assign them.) At each iteration, once all patch weights are computed, they are normalized so the sum of the weights stored at each vertex is 1. With this scheme, patch weights remain near 0.5 along patch boundaries, but smoothly fall to zero near the outer n-ring of the patch boundary.

Even with weights that sum to 1 at each vertex, the final tone conveyed by overlapping patterns can be altered in regions where patches are blended. (E.g., in a dark region requiring 100% ink application, using two halftone patterns that each outputs 50% ink will not achieve 100% ink coverage due to overlap in the patterns.)

We thus provide an additional control through which the user can increase or decrease the strength of the pattern around its boundaries. In the pixel shader, before the weight is used, it is raised to a power specified by the user – e.g., using a power of 0.5 increases the strength of the pattern moderately within the n-ring of the boundary. We often used this value in practice.

This method of computing weights and using them to blend between patches where they overlap effectively hides discontinuities between patches in many cases, and lets the user control the width of the overlap region. A drawback is that it depends on the mesh having a reasonably regular triangulation. More sophisticated schemes could be tried when that is not the case.

4 Results and discussion

The accompanying video shows our method in action. The results appear quite natural for camera or object motions that are well-approximated by a series of 2D similarity transforms. Such motions include camera pan or zoom, or camera rotation around the camera's forward line of sight.

The method produces mixed results for a single object rotating in front of the camera. For a compact object like a sphere, the 2D translation produced by the method is reasonable, since visible surfaces largely appear to translate in a consistent direction in 2D. The worst case (for static scenes) is when an elongated object rotates around an axis that is parallel to the film plane (see Figure 4). Looking straight on from the side, opposite ends of the object appear to move toward the center line. No 2D similarity transform well-approximates the apparent motion in this case.

We observed that while sliding can be a considerable problem for close examination of a single elongated shape, the problem is reduced when multiple objects are arranged in a scene (such as the landscape with cows, or the row of skulls shown in the video). Apparently the reason is that when navigating around several objects placed side by side, it is natural to move more slowly and gradually, compared to navigating around a single object. Sliding still occurs, but when sufficiently slowed, it is less noticeable.

Our method of weighting samples is designed to give higher priority to portions of surface that occupy more of the image. For comparison, we also implemented a simpler "unweighted" scheme that assigns weight = 1 if the sample is in the view frustum and front-facing, 0 otherwise. The results are generally similar, but the weighted scheme can be noticeably better, as in the case of the table shown in the video. In that example, the table top and support occupy little space in the image, and so are assigned small weights by the weighted scheme. That results in a better transform,

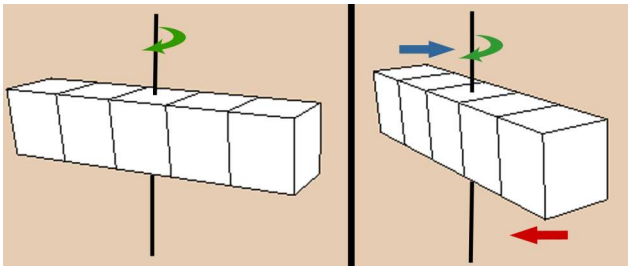

Figure 4: *Worst case scenario: an elongated object rotating around an axis parallel to the film plane. Opposite ends move toward the center line. Our method works poorly in this case since no 2D similarity transform well-approximates the apparent motion.*

Figure 5: *A painterly style applied to an animated model of a hand that has been divided into patches.*

because those surfaces move differently than the side of the table cloth, which fills most of the image and has a consistent motion.

We also observed a case when the weighted scheme seems worse than the unweighted one. The landscape model shown in the video has a large featureless foreground and a few hills on the horizon. The unweighted scheme assigns more importance to the distant hills, while the weighted scheme favors the foreground. Since the foreground lacks clear features, the viewer is less aware of sliding there, while the hills have a clearly perceived location, and any sliding of the pattern across them is quite noticeable. An interesting avenue for future work is thus to investigate weighting schemes that take into account more perceptual aspects of the scene.

A benefit of using 2D patterns is that they require no tedious surface parameterization, which might save time for artists. Artists are more free to design any pattern they want since patterns are ensured not to be deformed. Designs can easily be transferred to new objects. 2D patterns may also be better suitable for integration in hand-drawn animations.

We don't claim that 2D patterns should always be used in place of patterns that are mapped onto surfaces in 3D, especially in the case of foreground objects at the center of attention. Rather, we argue that 2D patterns will be attractive to designers and animators because they are easy to create, they perform well for many motions, and they provide a greater degree of visual abstraction. We envision interactive applications and animations that combine dynamic 2D patterns with other rendering techniques to produce both 2D and 3D effects with a unified artistic style.

References

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR 2006*, 141–149.

BRASSARD, L. 1998. *The Perception of the Image World*. PhD thesis, Simon Fraser University.

COCONU, L., DEUSSEN, O., AND HEGE, H.-C. 2006. Real-time pen-and-ink illustration of landscapes. In *NPAR 2006*, 27–35.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Proc. Graphics Interface*.

DURAND, F., OSTROMOUKHOV, V., MILLER, M., DURANLEAU, F., AND DORSEY, J. 2001. Decoupling strokes and high-level attributes for interactive traditional drawing. In *12th Eurographics Workshop on Rendering*, 71–82.

EISSELE, M., WEISKOPF, D., AND ERTL, T. 2004. Frame-to-Frame Coherent Halftoning in Image Space. In *Proceedings of Theory and Practice of Computer Graphics 2004*, 188–195.

FREUDENBERG, B., MASUCH, M., AND STROTHOTTE, T. 2002. Real-time halftoning: a primitive for non-photorealistic shading. In *13th Eurographics Workshop on Rendering*, 227–232.

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

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *SIGGRAPH 2000*, 517–526.

HORN, B. K. P. 1987. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America 4*, 4 (April), 629–642.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3, 755–762.

KAPLAN, M., AND COHEN, E. 2005. A generative model for dynamic canvas motion. In *Proceedings of Computational Aesthetics in Graphics, Visualization and Imaging*, 49–56.

LUFT, T., AND DEUSSEN, O. 2006. Real-time watercolor illustrations of plants using a blurred depth test. In *NPAR 2006*, 11–20.

MEIER, B. J. 1996. Painterly rendering for animation. In *SIGGRAPH 96*, 477–484.

NEHAB, D., AND SHILANE, P. 2004. Stratified point sampling of 3D models. In *Eurographics Symposium on Point-Based Graphics*, 49–56.

OSTROMOUKHOV, V. 1999. Digital facial engraving. In *SIGGRAPH 99*, 417–424.

ROST, R. J. 2006. *OpenGL Shading Language, Second Edition*. Addison Wesley Professional.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97*, 401–406.

WEBB, M., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2002. Fine tone control in hardware hatching. In *NPAR 2002*, 53–58.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *SIGGRAPH 94*, 91–100.

Acknowledgements

We thank Joe Sacco, Bill Plympton, and Richard Sala for permission to use details from their drawings in Figure 2, Igor Guskov for helpful advice in the early stages of this project, Haixiong Wang, Mike Cook and Rob Martinez for coding support, and the reviewers at INRIA and anonymous reviewers whose advice helped improve the paper. This research was supported in part by the NSF (CCF-0447883).