STAG: Smart Tools and Apps in computer Graphics (2016)
Giovanni Pintore and Filippo Stanco (Editors)

Tonal Art Maps with Image Space Strokes

László Szécsi[1] Marcell Szirányi[2] Ágota Kacsó[1]

[1]Budapest University of Technology and Economics
[2]Zen Studios

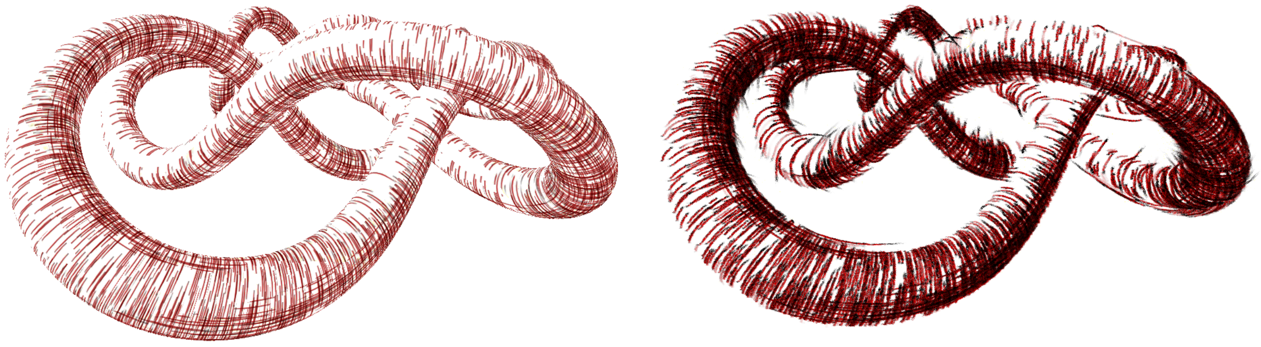

Figure 1: *Texture-space methods like TAM (left) must fade or clip strokes at object boundaries and for density control. TAMISS (right) restores stylistic coherence by fitting new strokes in image space.*

Abstract

This paper presents a hybrid hatching solution that uses robust and fast texture space hatching to gather stroke fragments, but fits stylized brush strokes over those fragments in image space. Thus we obtain a real-time solution that avoids the challenges associated with hidden stroke removal in image space approaches, but allows for the artistic stylization of strokes exceeding the limitations of texture space methods. This includes strokes running over outlines or behind occluders, uniquely random strokes, and adherence to image space brush properties.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Line and Curve Generation—

1. Introduction

Image synthesis methods mimicking artistic expression and illustration styles [Hae90, PHWF01, SS02] are usually vaguely classified as *non photo-realistic rendering* (NPR). Hatching is one of the basic artistic techniques that is often emulated in stylistic animation. Hatching strokes should appear hand-drawn, with roughly similar image-space width, dictated by pencil or brush size, but they should also stick to surfaces to provide proper object space shape and motion cues. Both properties must be maintained in an animation, without introducing temporal artifacts [JEGPO02, AWI*09]. In particular, when surface distance or viewing angle is changing, object-space density of strokes should adapt without the strokes flickering or drifting on the surface, while presenting natural randomness inherent in manual work. When zooming in or zooming out, this means adding and removing strokes gradually. Strokes should be uniform as drawn by the same brush or pencil, but also unique. Overdrawn lines crossing object contours often occur.

In this paper, we introduce a hybrid hatching approach built on texturing-based hatching, where we fit image-space strokes on hatching lines to provide style options not available just with surface shading.

2. Previous work

In pencil drawings artists convey the shape and illumination of objects with the density and orientation of thin hatch lines [WS94, HZ00]. To mimic this, we should define a *density* and a *direction field* in the image plane that is as close as possible to what an artist would use. Artists may apply more than one layer of parallel hatch lines, aligned at different angles to the direction field.

DOI: 10.2312/stag.20161362

This technique is called *cross-hatching*. The density should be influenced by the current illumination, while the direction field is determined either by the *principal curvature directions* [GIHL00] or the tone [LMLH07]. The principal curvature directions are those where the normal curvature of the surface has its maximum and minimum.

Several works proposed the application of *seeds* attached to objects [Mei96, USSK11]. Seeds are extruded to textured triangle strips representing hatching strokes in image space. Strokes are obtained by integrating the direction vector field, starting at seed points or particles [ZISS04, PBPS09]. The key problem in these methods is the generation of the world-space seed distribution corresponding to the desired image-space hatching density. This either means seed killing and fissioning [WH94]—even using mesh subdivision and simplification [CRL01]—, or rejection sampling [USSK11]. These techniques are mostly real-time, but require multiple passes and considerable resources. Compositing these with three-dimensional geometry is challenging: as extruded hatching curves do not strictly adhere to surfaces, depth testing them against triangle mesh objects must be using heavy bias and smooth rejection to avoid flickering.

In *texturing-based approaches*, hatches are generated into textures and mapped onto animated objects [LKL06] using an appropriate UV surface parametrization. This assures that strokes remain fixed to 3D surfaces, and the visibility problem is also solved robustly by conventional z-buffer depth testing of the textured surfaces. The most characteristic limitation of texturing-based hatching approaches is limited level-of-detail support. Simple static textures perform extremely poorly, as the width of hatching strokes is fixed in UV space, and—through the UV mapping—also in object space.

Thus, simple texturing does not allow for hatching that is uniform in screen space. A level-of-detail mechanism called *Tonal Art Maps* has been proposed to alleviate this problem [PHWF01]. Using this technique, several texture images are pre-drawn, representing different tones and hatching scales. Figure 2, taken from the referred paper, shows such a set of maps. When rendering surfaces, the appropriate texture in every pixel can be selected based on the desired tone and on-screen hatching stroke width. In order to avoid sharply clipped hatching strokes at boundaries of different-detail zones, the patterns fade into each other using interpolation.

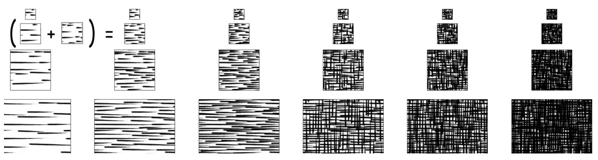

Figure 2: *A Tonal Art Map with the nesting property. Strokes in one image appear in all the images to the right and down from it. From Praun et. al [PHWF01].*

In an animation, as the required hatching density is changing, it is important that strokes stay at their on-surface positions. It is allowed for new hatching strokes to appear when the density increases, and for existing strokes to vanish, should the density decrease. However, strokes should not be appearing and vanishing in the same vicinity at the same time. Therefore, denser hatching textures should always contain the strokes of sparser textures as a subset. This is called the *nesting property*, also observable in figure 2. In the Tonal Art Maps method, direction of hatching on surfaces is defined by the UV mapping. Thus, the mapping needs to respect shape curvature characteristics. As this is challenging to achieve by a single mapping, *lapped textures* can be used.

Recursive Procedural Tonal Art Maps (RPTAM) [SS14] differ from TAM in that it does not use a static set of pre-drawn textures, Instead, it computes nearby stroke locations in texture space procedurally when shading a surface point. The stroke locations are generated so that they are nested in the same pattern repeated at half scale on a 2×2 grid. This allows for indefinite zooming onto surfaces.

Figure 3 shows stylistic artifacts that arise with both TAM and RPTAM as a result of their texture-space approach. In both methods, density control is performed on the pixel level. This means that the decision weather a stroke should appear can be different for different parts of a stroke. In we take a binary decision, some strokes are clipped before entering an area that should be less densely hatched. If the decision is smooth, e.g. implemented by blending between textures with different line densities, then strokes will fade out. Neither case is consistent with the requirement that the image is constructed using strokes matching and artists pencil or brush, in consistent style. Similarly, as strokes are applied to object surfaces in texture space, they are clipped at object silhouettes (or UV-parametrization discontinuities) in image space. This is not possible in hand-drawn art, and makes the renders appear artificial.

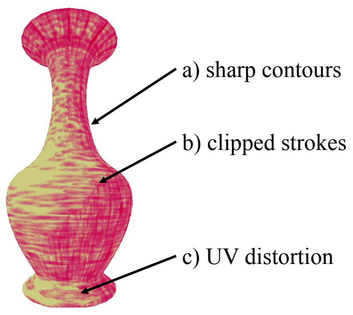

Figure 3: *TAM and RPTAM suffer from stylistic inconsistencies including: a) strokes clipped at object silhouettes, b) strokes clipped or faded for density control, and c) strokes distorted by anisotropic UV mapping.*

Fitting curves on outlines is a straightforward idea employed both for silhouettes [NM00] and in sketching [LFX*05]. Modeling hatching strokes as curves [KMM*02] was also used. We are not aware of any work where en masse curve fitting for hatching strokes was proposed.

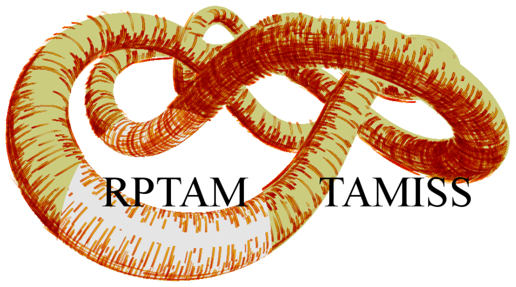

Figure 4: *TAMISS allows overdraw and eliminates clipping.*

3. Proposed method

In this paper we propose *Tonal Art Maps with Image Space Strokes* (TAMISS), a hybrid technique that combines the robust visibility testing and density control of TAM or RPTAM with the stylistic freedom of image space stroke extrusion (figure 4). The idea is to assign unique IDs to all TAM strokes, perform rasterization of surfaces with TAM, producing fragments marked with stroke IDs, and fit a curve on each set of fragments sharing the same ID. The curves can be extruded to image space strokes in proper style, while visibility and density control has already been taken care of by TAM.

3.1. Method outline

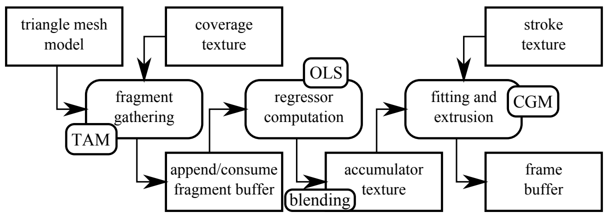

Figure 5: *Pipeline for the proposed method.*

Figure 5 depicts the algorithm workflow. We need TAM textures storing stroke IDs instead of colors. As multiple strokes may overlap, TAM texels need to contain short lists of stroke IDs. The RPTAM approach already makes use of such textures, called *stroke coverage textures* there. In the first, *fragment gathering* phase, surfaces are rasterized with TAM (see figure 6).

As a prerequisite, we need to identify all strokes with globally unique *stroke IDs*. This ID can be composed of an object index, a texture tile index, and the stroke's ID within the tile. The straightforward way to ensure that the IDs are unique is to use a non-overlapping UV mapping, i.e. an UV atlas. In case of overlapping mapping like lapped textures, the ID has to identify the layer the stroke appears on.

In the first phase of out proposed method, we render the scene geometry. In the fragment shader, as many stroke fragments are generated as there are strokes overlapping with the shaded surface point. All fragments, storing the globally unique stroke ID, are pushed into an appendable buffer. In the next, *summation* step, the fragments are aggregated by ID into descriptors. This is achieved by routing fragments by ID via *double hashing* into render targets with blending enabled. Finally, curves are fit on these descriptors, *extruded* to textured triangle strips, and rendered to the frame buffer.

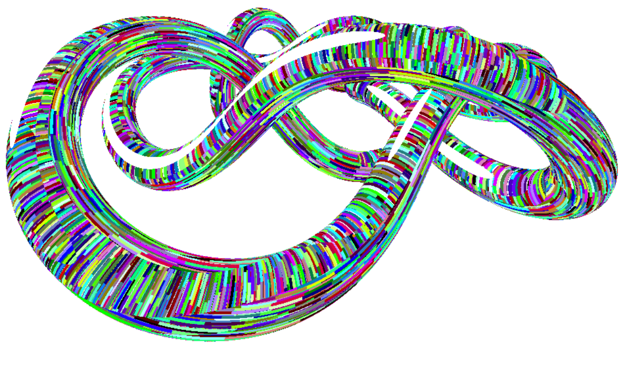

Figure 6: *A model rendered with TAM, outputting strokes IDs. Note that this image is never actually rendered in TAMISS, as we set no render target, when attempting to draw it. Instead, stroke fragments are gathered in a buffer.*

3.2. Regression

We rasterize the surfaces with appropriate TAM or RPTAM shaders to gather stroke fragments. Fragments are stored with their x_i, y_i screen space coordinates, and t_i values. Parameter t_i specifies where the fragment appears on the stroke. In RPTAM, it is available as the *stroke space coordinate*, for TAM, it must be stored in the coverage texture.

Given n fragments of a stroke, we need to find coefficients of the curve equation. We use cubic curves, so the parametric curve equation has the form $\mathbf{r}(t) = \left(\mathbf{c}_\mathrm{x}^\mathrm{T}\cdot\mathbf{t}, \mathbf{c}_\mathrm{y}^\mathrm{T}\cdot\mathbf{t}\right)$, with $\mathbf{t} = \left(1, t, t^2, t^3\right)^\mathrm{T}$, where $\mathbf{c}_\mathrm{x}$ and $\mathbf{c}_\mathrm{y}$ are column vectors of coefficients.

Finding $\mathbf{c}_\mathrm{x}$ and $\mathbf{c}_\mathrm{y}$ are *linear regression* problems, that we can solve using the *Ordinary Least Squares* method. Using the explicit formula by Hayashi [Hay00], we obtain the linear system

$$\mathbf{b} = \mathbf{A}\cdot\mathbf{c}, \text{ with } \mathbf{A} = \sum_{i=0}^{n-1} \mathbf{t}_i \cdot \mathbf{t}_i^\mathrm{T}, \quad \mathbf{t}_i = \left(1, t_i, t_i^2, t_i^3\right)^\mathrm{T}, \tag{1}$$

with either $\mathbf{b} = \sum_{i=0}^{n-1} \mathbf{t}_i \cdot x_i$ and $\mathbf{c} = \mathbf{c}_\mathrm{x}$, or $\mathbf{b} = \sum_{i=0}^{n-1} \mathbf{t}_i \cdot y_i$ and $\mathbf{c} = \mathbf{c}_\mathrm{y}$. Terms in sums correspond to stroke fragments. Thus, the terms to be summed for each fragment are $t_i^0, \ldots, t_i^6$, $t_i^0 \cdot x_i, \ldots, t_i^3 \cdot x_i$, and $t_i^0 \cdot y_i, \ldots, t_i^3 \cdot y_i$.

Solving the system for $\mathbf{c}$ by directly inverting $\mathbf{A}$ is feasible, but it is not the most efficient or stable option, as $\mathbf{A}$ may be singular or close to singular. As $\mathbf{A}$ is *positive definite*, the iterative *conjugate gradient method* [NW06] (CGM) can be applied, which delivers the pseudo-inverse solution even for singular matrices. In our experience, using 32-bit floating point numbers, performing the theoretically required four iterations is sufficient. Using solutions from previous frames as iteration starting points is therefore not worth the storage lookup time. Note that, in theory, any other method of solving the linear system is applicable, including *singular value* *decomposition* (SVD). However, CGM translates using only four multiplications of 4×4 matrices and a few four-element vector operations, making it efficient and easy to implement on the GPU.

The stroke may be only partially visible. We find the useful parameter range of the curve as

$$[t_{\min}, t_{\max}] = \left[\min_i t_i, \max_i t_i\right].$$

However, parts of strokes may be hidden, resulting in discontinuous fragment blocks. We assume that stroke visibility does not change more than two times within $[t_{\min}, t_{\max}]$. This gives us a third variable to perform regression on: visibility. In equation 1, $\mathbf{b} = \sum_{i=0}^{n-1} \mathbf{t} \cdot v_i$ and $\mathbf{c} = \mathbf{c}_\text{v}$, where visibility v_i is one at every fragment, and we have no data points for regression where v_i would be zero. Such data points, however, can easily be added by assuming they are evenly distributed in $[t_{\min}, t_{\max}]$. Given that v_i is zero at these points, $\mathbf{b}$ does not change, but $\mathbf{A}$ must include their contribution. For n new points, this can be computed analytically as

$$\mathbf{D} = n \int_{t_{\min}}^{t_{\max}} \mathbf{t} \cdot \mathbf{t}^\text{T} \mathrm{d}t, \text{ yielding } d_{i,j} = n \frac{t_{\max}^{i+j-1} - t_{\min}^{i+j-1}}{i+j-1}.$$

Solving the system $\mathbf{b} = (\mathbf{A} + \mathbf{D}) \cdot \mathbf{c}_\text{v}$, we obtain visibility function $v(t) = \mathbf{c}_\text{v}^\text{T} \cdot \mathbf{t}$, which can be thresholded to obtain visible stroke segments. A cubic fit on the visibility works as long as there are no more than two visible segments. More complex cases are rare, and easily pass as artistic inaccuracies. Extending the regression problem to a higher order fit is trivial, but a shader implementation would be much less elegant.

4. Implementation

The solid geometry depth is laid down first, so that only visible surfaces are rendered. The TAM or RPTAM implementation needs to be modified slightly to stream fragments into a buffer. This can be accomplished in a fragment shader without render target output, but writing to an appendable random access GPU buffer. This buffer of fragments can be rendered as a vertex buffer of point primitives. The vertex shader positions the point primitives by ID, sending them into a target buffer with blending. However, as IDs are global on all surfaces and detail levels, much more IDs are possible than the size of the target buffer. *Double hashing* can be used to map IDs to texels. To allow full parallelism, the hash table is read-only during a frame, but a new one is built by writing routed fragment IDs to an additional render target. When routing a fragment, the double hashing lookup is performed on the read-only table until either the fragment's stroke ID or an empty slot is found. In any case, the ID is written to the write-only hash table. For the next frame, the buffers are swapped. It is possible—if rare—that multiple newly appearing strokes try to claim the same empty slot, but that only means that some strokes are delayed by a frame.

After rasterization, blending is used to add $\mathbf{t}_i \cdot \mathbf{t}_i^\text{T}$, $\mathbf{t} \cdot x_i$, and $\mathbf{t} \cdot y_i$ to the texels. Values $t_{\min}$ and $t_{\max}$ are found with maximum blending.

Alternatively, without using the intermediate fragment buffer, the values can be aggregated using atomic operations. We found this adequate for lower order fitting, but as atomics only work

$\triangle$	$\sim$	OLS	CGM	E&R
27k	1k	3.06	0.04	0.11
27k	4k	2.96	0.19	0.32
90k	4k	2.86	0.31	2.01
193k	4k	2.78	0.23	0.51
193k	16k	2.88	0.91	2.04

Table 1: *Performance on different scenes (with the number of triangles and hatching strokes specified) at* 1920×1200*. Rendering times are given in ms for regressor aggregation (OLS), cubic curve fitting (CGM), and final stroke extrusion and rendering (E&R).*

with fixed-point representation, the floating point range provided by blending is indispensable for summing higher powers of t_i.

In the final pass, dataless point primitives are rendered for every texel of the aggregate texture. A geometry shader solves the regression equations using CGM, and extrudes the curve to a triangle strip in screen space. Any additional stylization like per-stroke randomization can be performed here.

4.1. Results and conclusions

We measured performance on an NVIDIA GeForce GTX 780, at 1920×1200 full-screen resolution (Table 4.1) using the knight and knot models (figure 7 and 8). Compared to single pass texturing with RPTAM, TAMISS takes about five times as long, but performs still well over 100 FPS in full screen. The most expensive operations are fragment aggregation, and final stroke rasterization. The solution of the regression equations with CGM is negligible. For more complex scenes, the overhead remains constant. Performance depends heavily on the number of strokes, but 32k strokes in a frame are always more than sufficient.

As fitting is performed independently in every frame, any rasterization or numerical inaccuracies may manifest as jittering in stroke positions. This can be alleviated by using a higher resolution for fragment gathering, resulting in more fragments to process. This cost could be amortized by averaging aggregate fragment data over several frames instead.

While higher order fitting is certainly possible, even a quartic solution would need approximately doubled computation and storage requirements. We think this would be for no practical gain, as the underlying shape is adequately represented by cubics. This does not mean that strokes could not have additional stylization like waves or zigzags, even on a stroke-by-stroke basis. Thus, our method does not limit visual style compared to TAM, but allows for more stylistic freedom by decoupling stroke positioning and stroke style.

The main limitation of our method is that it needs an overlap-free UV mapping, oriented on object features. Such an UV mapping is never readily available, and we have yet to propose an automated solution, or show that an existing one like *lapped textures* as in [PHWF01] can be adapted.

Acknowledgements

This work has been supported by OTKA PD-104710 and OTKA K-104476.

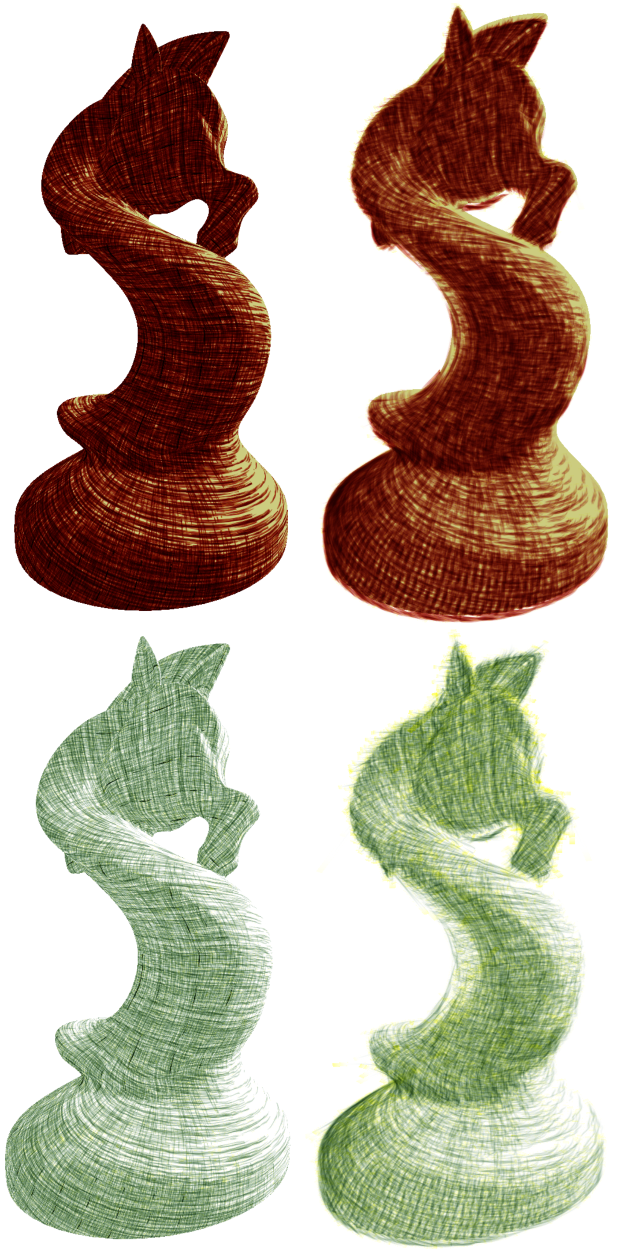

Figure 7: *Knight model rendered with RPTAM (left) and TAMISS (right).*

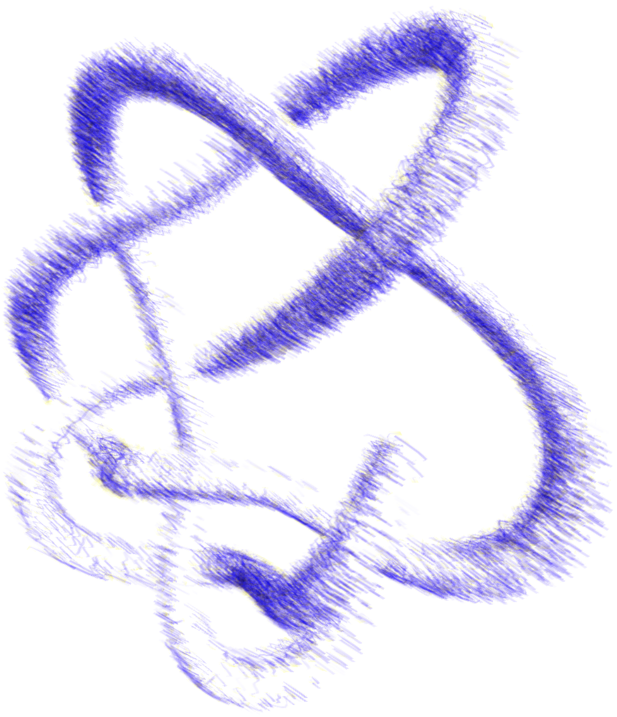

Figure 8: *Torus knot model rendered in a pen-like style. All lines are uniquely randomized.*

References

[AWI*09] ALMERAJ Z., WYVILL B., ISENBERG T., GOOCH A. A., GUY R.: Automatically mimicking unique hand-drawn pencil lines. *Computers & Graphics 33*, 4 (2009), 496–508. 1

[CRL01] CORNISH D., ROWAN A., LUEBKE D.: View-dependent particles for interactive non-photorealistic rendering. In *Graphics interface* (2001), vol. 1, pp. 151–158. 2

[GIHL00] GIRSCHICK A., INTERRANTE V., HAKER S., LEMOINE T.: Line direction matters: an argument for the use of principal directions in 3D line drawings. In *International Symposium on Non-Photorealistic Animation and Rendering 2000* (2000), pp. 43–52. 2

[Hae90] HAEBERLI P.: Paint by numbers: Abstract image representations. In *ACM SIGGRAPH Computer Graphics* (1990), vol. 24, ACM, pp. 207–214. 1

[Hay00] HAYASHI F.: *Econometrics*. Princeton University Press, 2000. 3

[HZ00] HERZMANN A., ZORIN D.: Illustrating smooth surfaces. In *ACM SIGGRAPH 2000* (2000), pp. 433–438. 1

[JEGPO02] JODOIN P.-M., EPSTEIN E., GRANGER-PICHÉ M., OSTROMOUKHOV V.: Hatching by example: a statistical approach. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (2002), ACM, pp. 29–36. 1

[KMM*02] KALNINS R., MARKOSIAN L., MEIER B., KOWALSKI M., LEE J., DAVIDSON P., WEBB M., HUGHES J., FINKELSTEIN A.: Wysiwyg npr: Drawing strokes directly on 3d models. *ACM Transactions on Graphics 21*, 3 (2002), 755–762. 2

[LFX*05] LEWIS J. P., FONG N., XUEXIANG X., SOON S. H., FENG T.: More optimal strokes for npr sketching. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2005), ACM, pp. 47–50. 2

[LKL06] LEE H., KWON S., LEE S.: Real-time pencil rendering. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (2006), ACM, pp. 37–45. 2

[LMLH07] LEE Y., MARKOSIAN L., LEE S., HUGHES J. F.: Line drawings via abstracted shading. In *SIGGRAPH '07* (2007), p. 18. doi:http://doi.acm.org/10.1145/1275808.1276400. 2

[Mei96] MEIER B. J.: Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM, pp. 477–484. 2

[NM00] Northrup J., Markosian L.: Artistic silhouettes: A hybrid approach. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering* (2000), ACM, pp. 31–37. 2

[NW06] Nocedal J., Wright S. J.: *Conjugate gradient methods*. Springer, 2006. 3

[PBPS09] Paiva A., Brazil E. V., Petronetto F., Sousa M. C.: Fluid-based hatching for tone mapping in line illustrations. *The Visual Computer 25*, 5-7 (2009), 519–527. 2

[PHWF01] Praun E., Hoppe H., Webb M., Finkelstein A.: Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 581–581. 1, 2, 4

[SS02] Strothotte T., Schlechtweg S.: *Non-photorealistic computer graphics: modeling, rendering, and animation*. Elsevier, 2002. 1

[SS14] Szécsi L., Szirányi M.: Recursive procedural tonal art maps. In *WSCG 2014 Full Papers Proceedings* (2014), Union Agency, pp. 57–66. 2

[USSK11] Umenhoffer T., Szécsi L., Szirmay-Kalos L.: Hatching for motion picture production. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 533–542. 2

[WH94] Witkin A. P., Heckbert P. S.: Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), ACM, pp. 269–277. 2

[WS94] Winkenback G., Salesin D. H.: Computer generated pen-and-ink illustration. In *ACM SIGGRAPH 94* (1994), pp. 91–100. 1

[ZISS04] Zander J., Isenberg T., Schlechtweg S., Strothotte T.: High quality hatching. In *Computer Graphics Forum* (2004), vol. 23, Wiley Online Library, pp. 421–430. 2