Tonal Art Maps with Image Space Strokes

László Szécsi, Marcell Szirányi and Ágota Kacsó

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 and previous work

Hatching is one of the basic artistic techniques that is often emulated in stylistic animation. Strokes should be uniform as drawn by the same brush or pencil, but also unique. Overdrawn lines crossing object contours often occur.

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. 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.

The visibility problem is solved robustly in *texturing-based approaches* [LKL06], where the hatching pattern is drawn on object surfaces. Uniform screen-space hatching density can be achieved using different level-of-detail textures, either pre-drawn as *Tonal Art Maps (TAM)* [PHWF01], or procedurally defined as *Recursive Procedural Tonal Art Maps (RPTAM)* [SS14]. Deviating from a hand-drawn style, strokes can appear clipped when crossing over to less densely hatched areas, or at object silhouettes and parametrization discontinuities.

2. 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 1). 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.

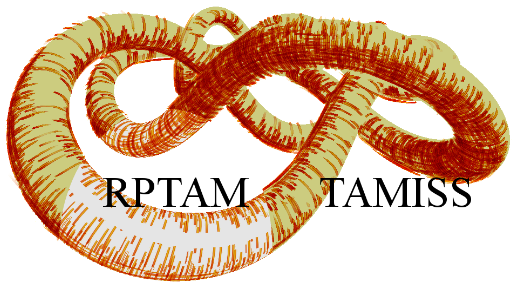

Figure 1: *TAMISS allows overdraw and eliminates clipping.*

2.1. Method outline

Figure 2 depicts the algorithm workflow. We need TAM textures storing stroke IDs instead of colors. In the first, *fragment gathering* phase, surfaces are rasterized with TAM. For every surface fragment, as many stroke fragments are generated as there are overlapping strokes. All fragments store a globally unique *stroke ID*. In the next, *summation* step, the fragments are aggregated by ID into descriptors. Finally, curves are fit on these descriptors, *extruded* to textured triangle strips, and rendered to the frame buffer.

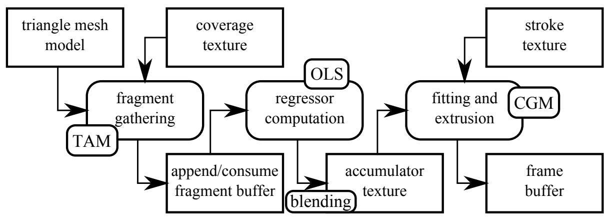

Figure 2: *Pipeline for the proposed method.*

DOI: 10.2312/egp.20161046

2.2. 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.

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 $\mathbf{r}(t) = \left(\mathbf{c}_x^T \cdot \mathbf{t}, \mathbf{c}_y^T \cdot \mathbf{t}\right)$, with $\mathbf{t} = \left(1, t, t^2, t^3\right)^T$, where $\mathbf{c}_x$ and $\mathbf{c}_y$ are column vectors of coefficients.

Finding $\mathbf{c}_x$ and $\mathbf{c}_y$ 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

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

with either $\mathbf{b} = \sum_{i=0}^{n-1} \mathbf{t}_i \cdot x_i$ and $\mathbf{c} = \mathbf{c}_x$, or $\mathbf{b} = \sum_{i=0}^{n-1} \mathbf{t}_i \cdot y_i$ and $\mathbf{c} = \mathbf{c}_y$. Terms in sums correspond to stroke fragments.

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

3. 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 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. 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 $\mathbf{t}_i \cdot \mathbf{t}_i^T$, $\mathbf{t} \cdot x_i$, and $\mathbf{t} \cdot y_i$ to the texels. Values $t_{\min}$ and $t_{\max}$ are found with maximum blending.

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.

3.1. Results and conclusions

We measured performance on an NVIDIA GeForce GTX 780, at 1920×1200 full-screen resolution (Table 3.1). 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 sufficient.

△	∼	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).*

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.

References

[Hay00] Hayashi F.: *Econometrics*. Princeton University Press, 2000. 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. 1

[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. 1

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

[SS14] Szécsi L., Szirányi M.: Recursive porcedural tonal art maps. In *WSCG 2014 Full Papers Proceedings* (2014), Union Agency, pp. 57–66. 1

[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. 1