Procedural band patterns

Jimmy Etienne, Sylvain Lefebvre

Procedural band patterns

Jimmy Etienne
Université de Lorraine, CNRS, Inria, LORIA
jimmy.etienne@inria.fr

Sylvain Lefebvre
Université de Lorraine, CNRS, Inria, LORIA
sylvain.lefebvre@inria.fr

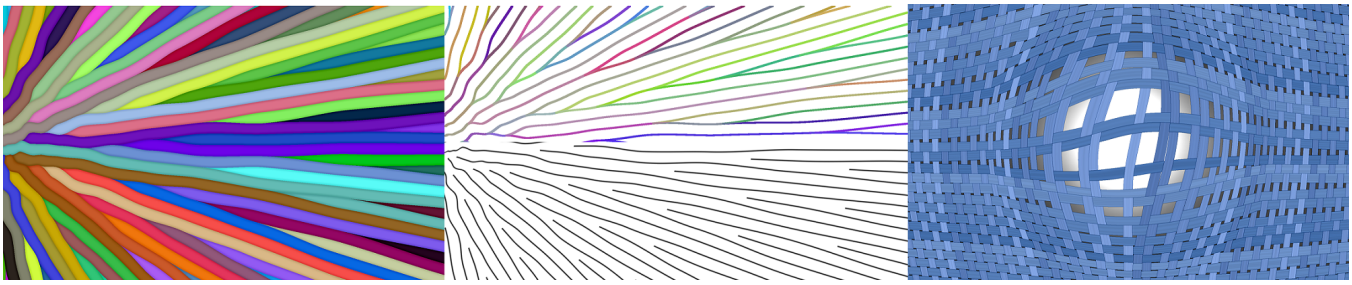


Figure 1: Our technique generates band patterns following a parametric field, while adapting the density of bands per unit. Each band is uniquely identified (*left*) which affords for robust extraction of border trajectories and center lines (*middle*). The method is procedural which allows a wide range of dynamic shader effects, such as textile patterns that adapts to stretch (*right*). Contrary to classical subdivision approaches, our approach introduces new bands with a non power of two factor, allowing a more progressive gradation.

ABSTRACT

We seek to cover a parametric domain with a set of evenly spaced bands which number and width varies according to a density field. We propose an implicit procedural algorithm, that generates the band pattern from a pixel shader and adapts to changes to the control fields in real time. Each band is uniquely identified by an integer. This allows a wide range of texturing effects, including specifying a different appearance in each individual bands. Our technique also affords for progressive gradations of scales, avoiding the abrupt doubling of the number of lines of typical subdivision approaches. This leads to a general approach for drawing bands, drawing splitting and merging curves, and drawing evenly spaced streamlines. Using these base ingredients, we demonstrate a wide variety of texturing effects.

CCS CONCEPTS

• **Computing methodologies** → **Texturing**.

ACM Reference Format:
Jimmy Etienne and Sylvain Lefebvre. 2020. Procedural band patterns. In *Symposium on Interactive 3D Graphics and Games (I3D '20), May 5–7, 2020, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3384382.3384522

1 INTRODUCTION

We focus on producing patterns of parallel bands along a given parametric field, while locally adapting the number of bands to an input control field. More precisely, we expect as input a domain $\Omega \in \mathbb{R}^2$ or $\Omega \in \mathbb{R}^3$, a parameter field $u : \Omega \rightarrow \mathbb{R}$, and a density field $d : \Omega \rightarrow \mathbb{R}$ that defines a target number of bands per unit of u. These fields may be the direct result of an optimization [Groen et al. 2019] or may be painted by the user. The field d may be linked to u (e.g. compensating for a distortion) or may be independent from it, in which case the actual local density of bands will be a combination of the intrinsic stretch of u and the density factor of d. Both u and d are assumed to be continuous, smooth scalar fields.

We denote by p a point in Ω, such that we obtain the parameter value $u(p)$ and density value $d(p)$ at each point p. The objective is to produce a set of bands which are flowing along the isovalues of u and where the local density of bands – or equivalently their spacing – is changing according to d.

The originality of our approach is to define a lookup function $\mathcal{B}\,(v, s) : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{N}$ that returns the unique integer ID identifying the band enclosing parameter value v, and where the local density is controlled by s. In our context the lookup is performed for a point $p \in \Omega$ as $\mathcal{B}\,(u(p), d(p))$. $\mathcal{B}$ is *procedural* [Lagae et al. 2010]: it has a minimal computational and memory requirement, allowing its implementation in a pixel shader for fast synthesis and interactive manipulation and animation of complex patterns. The returned integers uniquely identify the bands allowing for a wide range of texturing effects.

Previous work Drawing evenly spaced curves and streamlines is a long standing problem in Computer Graphics. We identify three main families: approaches tracing streamlines in a vector field from starting points [Hertzmann and Zorin 2000; Jobard and Lefer 1997; Mebarki et al. 2005; Spencer et al. 2009], approaches based on global periodic parameterizations [Knöppel et al. 2015] and finally approaches splitting and merging covering curves, as pioneered by the seminal work of Elber and Cohen [Elber and Cohen 1996]. Our

I3D '20, May 5–7, 2020, San Francisco, CA, USA

ACM ISBN 978-1-4503-7589-4/20/05…$15.00
https://doi.org/10.1145/3384382.3384522

approach most closely relates to this later family, and in particular its image space variants [Groen et al. 2019].

Contributions. Our approach introduces several improvements. First, by defining numbered bands as opposed to directly curves, we produce a partition of the domain Ω, enabling a robust (and simple!) extraction of each region and its boundary. The unique identifier also affords for direct manipulation of the bands and their borders, for texturing and patterning effects. For instance, drawing the center line of fully deployed bands produces evenly spaced streamlines. Second, existing subdivision techniques define gradation by doubling the density. This leads to abrupt jumps in the number of curves and provides only a crude approximation of the target scale. Instead, our technique allows a finer control. Finally, our approach is fully implicit and defined by a procedural lookup function, avoiding global geometric constructions and optimization. We are not aware of any other technique offering a similar capability.

2 METHOD

Given the fields d and u, we cover the domain Ω with a discrete grid and synthesize the bands by evaluating $\mathcal{B}$ on every node. This is implemented as a pixel shader (GLSL) applied to a render target. The code is given in Section 4. The remainder of the text progressively introduces the principles and elements of the algorithm.

2.1 Overview

Given a spacing s, it is trivial to produce parallel bands in $u(p)$ by defining the band identifier as $\lfloor \frac{u(p)}{s} \rfloor$. The principle of our approach is to cover the domain with many overlapping sets of parallel bands, using different spacings. The spacings are decreasing powers of a base value $1 < step \leq 2$. We call *bands of level L* the set using a spacing of $step^{-L}$. The choice of $step$ allows a trade-off between a doubling split ($step = 2$) or a more progressive gradation, which we discuss in Section 3.

We select which set should appear in a given location following $d(p)$, and interpolate (deform) the bands, with smaller bands borders joining or closing onto coarser band borders. This in turn defines a parent–child relationship which we use to define a global numbering through the band hierarchy.

2.2 Procedural bands

Given a point p, we seek to compute the global identifier of the band enclosing p. This is done in two steps. First we identify the local ID of the band to which p belongs at the density $d(p)$. The local ID is the rank of the band in its set, it is computed by function `getLocalID` (see code Section 4). Second, we produce a global ID for the band in function `getGlobalID`.

We now focus on `getLocalID` (line 5). We first determine from $d(p)$ the band levels bracketing the target density. This is achieved by quantizing $d(p)$ based on the value of $step$, see the `quantize` function line 1. This gives the situation illustrated in Figure 2: p belongs to two bands, one in a set having more bands than the target $d(p)$, and one in a set having fewer bands.

We then define an interpolation that displaces the band borders, making the finer bands deform towards their coarser counterparts. This is illustrated in Figure 3. The interpolation is performed by pulling each finer band border towards its closest coarser band

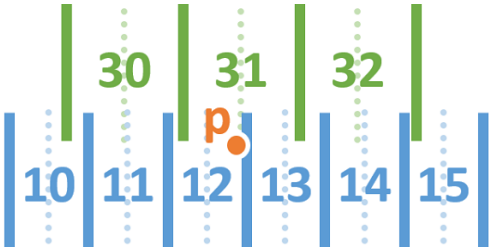


Figure 2: Every point p belongs to a band in a finer and a coarser set enclosing the target density $d(p)$: the finer set has more bands than desired at $d(p)$ while the coarser set has fewer bands.

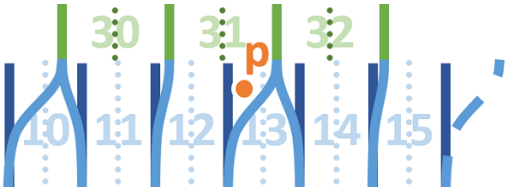


Figure 3: The borders of the finer bands are pulled towards their closest coarser band border. A point p in a transition area may end up in a different band after interpolation.

border. The interpolator is given by the position of $d(p)$ within the density interval of the bracketing levels. Note that as we change the location of the band boundaries, p may no longer be in the same initial band. This is tested and adjusted for lines 23 and 24. Through interpolation, some finer bands are *closed*: both their borders move towards the same coarser band border, e.g. band 13 in Figure 3.

After this process, we obtain the ID of the band enclosing p in the level just finer than $d(p)$. This ID is local to this particular level: if used directly, it would change arbitrarily where a change of level occurs in d. Instead, we produce a global ID, valid *across* levels. This is achieved in `getGlobalID` (line 28). Note that since we have $step \leq 2$, we at most double the number of bands from one level to the next. Thus, our approach is to number the bands in the same way we number leaves in a binary tree. From the current band level, we move up the hierarchy (lines 32–52), and insert a bit in the ID (line 43) each time the band closes (lines 35–41). Finally, the ID of the top level band is added as the most significant bits of the global ID (line 54). This is illustrated in Figure 4. In other words, the global ID is the ID of the top level band, followed by the binary pattern of opening bands to reach the band enclosing p.

As an added bonus, we can easily detect when a band *just appears*, by checking if the first band is closing immediately (lines 44–47). This is an interesting capability, as such a band is one that is smaller than the target spacing – there is not yet enough space between the parents for it to be fully deployed. In particular, we may choose to remove such a band or draw it in a different style.

Band shifting Using an irregular subdivision pattern ($step < 2$) poses an interesting challenge. Around the origin, all band sets align, producing an undesirable pattern shown in Figure 5, left.

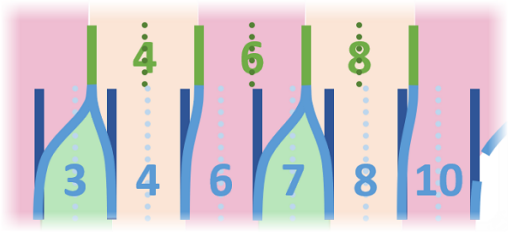


Figure 4: Band global IDs are computed in a hierarchical fashion, with parent defining most significant bits while children use least significant ones. The ID of closing bands no longer appears in lower levels, leaving a gap between IDs.

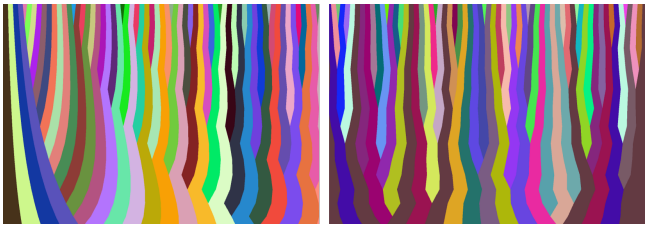


Figure 5: *Left:* Without shifts, the alignment of all band sets at the origin produce an artificial, peculiar pattern. *Right:* The random shifts break the alignment, making the pattern similar everywhere.

This can be suppressed by translating the band sets by a (pseudo-)random shift, which is different for each band level. The effect is shown in Figure 5, right.

For $step = 2$, we shift the bands at every level by half their spacing to obtain the traditional balanced subdivision pattern.

3 CONTROLS AND PARAMETERS

Progressive increase in density By changing *step*, we control the number of bands opening at every level. For $step = 2$ our technique behaves similarly to traditional subdivision approaches, with one new band opening for each coarser band, effectively doubling the number of bands at every level.

It is worth outlining that this approach, while very typical, also gives only a crude approximation of the target density. Indeed, across an interpolation range the produced density (number of bands) is wrong by as much as 50% (in the middle).

Using lower values of *step*, our technique affords for a very progressive insertion of bands at every new levels. Let us consider a value $step = \frac{N}{M}$ with $M, N \in \mathbb{N}$ forming an irreducible fraction, and $M < N \leq 2M$. Such a step will open $N - M$ new bands for every M bands in the previous level – this stems from the fact that each level L corresponds to a density $step^L$.

$N = 2,\ M = 1$ gives the standard case (100% increase every level). $N = 3,\ M = 2$ opens one new band every two coarser bands (50% increase). $N = 17,\ M = 13$ opens four new bands every thirteen coarser ones (31% increase). These number can be freely chosen as long as *step* remains in the $]1, 2]$ interval.

Regularity and periodicity The produced pattern is entirely defined by the subdivision of the top level bands (for $d = step$, we have one band per unit in u). It is intriguing to consider the periodicity of the split pattern. For the sake of clarity we ignore the random shifts in this discussion and assume all band sets are aligned on the origin.

Let us again write *step* as an irreducible fraction $\frac{N}{M}$. From one level to the next, a period occurs in the split pattern every M bands, as borders across both levels align: this repeats the pattern at the origin. However, this is the case for *two consecutive* levels. When we consider more levels, the period before all borders align grows. In fact, for k levels the period becomes M^k. Indeed, given Q bands at the base level, the number of bands at level k is $Q\left(\frac{N}{M}\right)^k$. This number will only be an integer value for $Q = M^k$, since M and N are coprime (irreducible fraction).

Interpolation profile All our results use a simple linear interpolation. However, using a different profile (e.g. GLSL smoothstep) leads to different (smoother) band shapes in the opening region. In addition, the interpolation profile can control how 'quickly' new bands open to their full width.

4 PSEUDO-CODE

The detailed pseudo-code is given next. We also refer the reader to our shadertoy implementations (see Table 1).

```
1  float quantize(float d) {
2    return pow(step, floor(log(d)/log(step)));
3  }
4
5  int getLocalID(float u, float d) {
6    // fine and coarse densities
7    int level = d2level(d);
8    float df  = level2d(level  );
9    float dc  = level2d(level-1);
10   // get enclosing band id
11   int lid   = u2id(u, df);
12   // get the borders of the finer band
13   float lf_brdr = id2Lborder(lid  , df);
14   float rf_brdr = id2Lborder(lid+1, df);
15   // get the closest coarser band borders
16   float lc_brdr = u2closestBorder(lf_brdr, dc);
17   float rc_brdr = u2closestBorder(rf_brdr, dc);
18   // interpolate between fine and coarse borders
19   float a     = interpolate(d);
20   float left  = lf_brdr * a + lc_brdr * (1.0 - a);
21   float right = rf_brdr * a + rc_brdr * (1.0 - a);
22   // adjust enclosing band after interpolation
23   if (u < left ) lid--;
24   if (u > right) lid++;
25   return lid;
26 }
27
28 uvec4 getGlobalID(int lid, float d) {
29   uvec4 gid = uvec4(0u);
30   int start_level = d2level(d);
31   for (int level = start_level; level > 0
32                               ; level--) {
33     float df      = level2d(level);
34     float dc      = level2d(level-1);
35     // get the borders of the fine band
36     float lf_brdr = id2Lborder(lid  , df);
37     float rf_brdr = id2Lborder(lid+1, df);
38     // get the closest coarser band borders
39     float lc_brdr = u2closestBorder(lf_brdr, dc);
40     float rc_brdr = u2closestBorder(rf_brdr, dc);
41     if (lid_brdr == rid_brdr) {
42       // this band is closing between parents
43       gid = setIDBit(gid, uint(96 - level));
44       lid  = u2id(lf_brdr, dc);
45       if (level == start_level) {
46         // the start band is appearing
47       }
48     } else {
49       // move up to parent
50       lid  = lid_brdr;
51     }
52   }
53   // parent ids into most significant 32 bits
54   gid = setMostSignificantBits(gid,uint(lid));
55   return gid;
56 }
```

5 RESULTS

Our technique opens multiple possibilities to produce band and curve patterns. In this section we first show some basic patterns obtained directly from our technique, and discuss the effect of parameters in Section 5.1. We then discuss how to use various properties to produce more elaborate animated patterns in Section 5.2. Finally, we discuss how trajectory extraction can be used to produce graded fill patterns for additive manufacturing in Section 5.3.

5.1 Bands and curves

Figure 6 shows two complex cases of fields u and d with bands produced for various values of *step*. The left most is $step = 2$ while the others are showing decreasing values ($step = \frac{17}{13}, step = \frac{79}{71}$). The parts below the dashed lines are showing the interpolation regions (discretization of d). Note how as *step* decreases, the discretization of d refines, producing a better match to the input field. For small values of *step*, the distribution of splits becomes less regular but the bands are also more 'jaggy'. A suitable tradeoff is easily found by interactive manipulation.

Figure 1 shows three drawing modes enabled by our approach. The first is to extract a network of splitting/merging curves from the boundaries of the band regions. Here, the band ids allow to robustly decide which pixel edges belong to a curve, and also provide an ID for each extracted trajectory (the pair of band IDs on either side). The second is to draw the middle lines of fully deployed bands (see test lines 44–47) to obtain evenly spaces streamlines. The third is to draw bands directly, here with a subtle shadowing effect. Much more is possible in terms of patterning, as we will discuss next.

5.2 Animated patterns

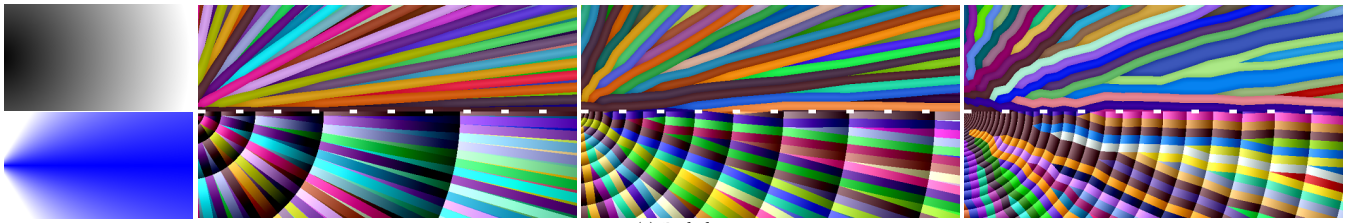
Note: These effects are best seen animated. We provide the shadertoy links at the end of this section (Table 1).

We provide examples of animated effects, exploiting several unique possibilities. We generate the effects by having procedural fields u and d dynamically drive one or multiple band patterns. The u field is obtained analytically (sum of sines, radial basis functions and noise), while d is often computed from the gradient of u to compensate for distortion. We then use band IDs as well as the ability to compute a local parameterization (distance of lookup point to enclosing band left border) to achieve various effects.

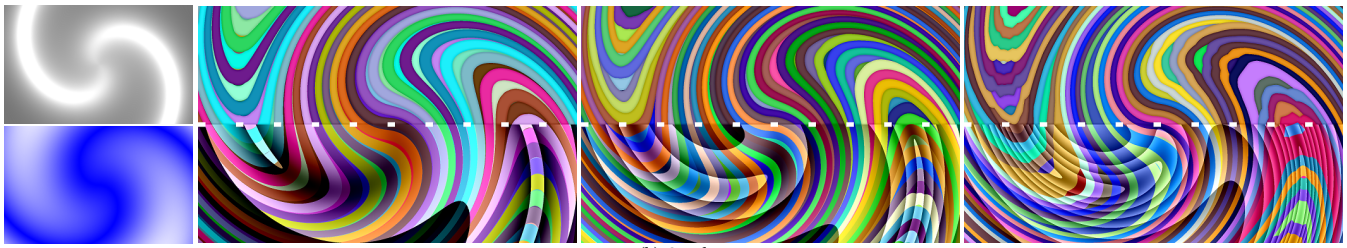
Simple yet interesting effects are obtained by coloring the bands in a stretching field with distortion compensation, see for instance *flow bands*. Driving the band density also produces interesting results, such as in *scaling bands* where the band density is increased in a moving circular area. The *band flag* effect combines two sets of adapting bands, with a coloring that retains consistency along each direction, producing a typical towel coloring scheme.

We also explored tearing animations, where we discard all lines appearing in between the main parent lines. This is easily done by checking the least significant bits of the band IDs. This produces an effect where the main bands maintain their width, while spaces appear in between. The parameter *step* then controls how and whether the bands form groups under stretch. The shaders *tear twist, lens, light claws, textile* and *net* are obtained in this manner. Instead of discarding, we can also change the coloring of the bands, producing effects such as the *hairdryer* shader. Here, new darker hair strands appear as the motion expands the textured area.

The combination of two sets of adapting bands and tearing leads to effect producing squares, for instance in *lens* and *light claws*. The *lens* effect is a lens where instead of zooming the squares maintain their size and the bands tear to accommodate for the distortion. Note the uneven split pattern thanks to a choice of $step < 2$. The *light claws* effect drives a distortion field from sound, producing dynamically splitting bands along lines in two directions, disconnecting squares.



(a) Radial



(b) Swirl

Figure 6: Two results using complex u and d fields, using $step = 2$, $\frac{17}{13}$, $\frac{79}{71}$. Left: u field in blue-white, d in shades of gray. Below dashed line: discretization steps of d overlaid, the more steps the more precise is the fit between the actual and the target number of bands.

Table 1: Links to anonymized shaders.

Flow bands	https://www.shadertoy.com/view/wlt3DM
Scaling bands	https://www.shadertoy.com/view/tlt3DM
Tear twist	https://www.shadertoy.com/view/WttGD8
Band flag	https://www.shadertoy.com/view/3ttGD8
Textile	https://www.shadertoy.com/view/wl3GDH
Net	https://www.shadertoy.com/view/3ldGD8
Lens	https://www.shadertoy.com/view/WltGD8
Light claws	https://www.shadertoy.com/view/tt33W7
Hairdryer	https://www.shadertoy.com/view/3tdGW8

We can also weave two orthogonal sets of bands to produce textiles, for instance in *net* and *textile*. We determine which band is front at crossings using binary checks on their IDs. We also the ability to compute a local parameterization within the bands width to make then thinner. Note how under distortion the weaving expands and slides while the bands maintain their width, revealing the background.

All these shaders are dynamic and real time – thus they would work in a rendering context, for instance to adapt a texture to stretch. As our technique only requires a single input coordinate (u), it can also be applied to solids and implicit surfaces.

5.3 Infill patterns for 3D printing

We use our technique to produce fill patterns inspired from so-called cubic infills [Lefebvre 2015; Wu et al. 2016]. We extract the trajectories as the contours between pixels associated with different band IDs. This process is fast and robust – akin to extracting the contours of a white region in a binary image. The key advantage is that our technique allows for spatial grading of density while producing very continuous paths. This has been implemented (and shipping to users) in our slicer [Sylvain Lefebvre 2017] for more than two years, and was in fact the initial motivation behind this work.

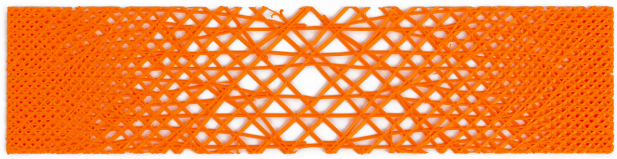


Figure 7: 3D printed part revealing an infill pattern obtained from our technique. Note the very progressive change in density and the curve network extracted at each layer.

There are other contexts in additive manufacturing where iso-lines of equal spacing are desirable, for instance to generate fill patterns following optimized fields [Steuben et al. 2016] or for curved 3D printing [Ezair et al. 2018]. We believe our approach to be especially promising to decompose a 3D shape into solid slabs, before filling them with curved paths (using contouring and zigzag fill within the extracted curved slab [Chakraborty et al. 2008]).

6 LIMITATIONS

Our technique has a number of limitations, that can be problematic depending on the intended use.

In areas where the density field d varies quickly, bands will appear distorted. This is for instance visible in Figure 6, bottom right, in areas of high curvature. It would be interesting to apply filtering to d such as to limit such defects.

Intermediate density levels always exhibit partially deployed bands. A consequence is that, while the number of bands remains correct, their spacing may look uneven in areas of constant scale

(linear u and constant d). This situation may be minimized – albeit not entirely resolved – by modifying the interpolation profile to quickly open/close bands (see Section 3).

In general we believe it would be desirable to randomize the locations where the band splits occur. While our irregular splitting strategy ($step < 2$) reduces split alignments, they remain located along isolines of d and not randomly distributed.

The numbering scheme (`getGlobalID`) relies on integers and might run out of numbers, being unable to distinguish between bands deep in the hierarchy. Using wider integers is not too difficult in our context as the only operation is to set bits in the IDs. Nevertheless infinite zooming would require recycling IDs.

7 CONCLUSION

We propose a fully procedural technique to generate band patterns. It is implemented as a fast pixel shader that interactively reacts to changes to the control fields. We believe our technique to be useful for a wide range of applications, such as visualization, texturing, but also for producing fill patterns in additive manufacturing.

Our technique extends trivially to 3D, for instance allowing to decompose a shape into solid slabs – the equivalent of our bands – while retaining all the procedural and adaptive capabilities of our technique. We envision direct applications in modeling of solid properties and process planing for additive manufacturing.

8 ACKNOWLEDGEMENTS

The work was partly supported by Région Grand-Est, Lorraine Université d'Excellence (ANR-15-IDEX-04-LUE) and CNRS. We thank Cédric Zanni for proof reading.

REFERENCES

Debapriya Chakraborty, B. Aneesh Reddy, and A. Roy Choudhury. 2008. Extruder Path Generation for Curved Layer Fused Deposition Modeling. *Comput. Aided Des.* 40, 2 (2008), 235–243.

Gershon Elber and Elaine Cohen. 1996. Adaptive Isocurve-based Rendering for Freeform Surfaces. *ACM Transactions on Graphics* 15, 3 (1996), 249–263.

Ben Ezair, Saul Fuhrmann, and Gershon Elber. 2018. Volumetric covering print-paths for additive manufacturing of 3D models. *Computer-Aided Design* 100 (2018), 1–13.

Jeroen P Groen, Jun Wu, and Ole Sigmund. 2019. Homogenization-based stiffness optimization and projection of 2D coated structures with orthotropic infill. *Computer Methods in Applied Mechanics and Engineering* 349 (2019), 722–742.

Aaron Hertzmann and Denis Zorin. 2000. Illustrating Smooth Surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*. 517–526.

Bruno Jobard and Wilfrid Lefer. 1997. Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Visualization in Scientific Computing '97*. Springer Vienna, 43–55.

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Transactions on Graphics* 34 (2015). Issue 4.

A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, and M. Zwicker. 2010. A Survey of Procedural Noise Functions. *Computer Graphics Forum* (2010). https://doi.org/10.1111/j.1467-8659.2010.01827.x

Sylvain Lefebvre. 2015. 3d infilling: faster, stronger, simpler. http://sylefeb.blogspot.fr/2015/07/3dprint-3d-infilling-faster-stronger.html.

A. Mebarki, P. Alliez, and O. Devillers. 2005. Farthest point seeding for efficient placement of streamlines. In *IEEE Visualization, 2005.* 479–486.

Benjamin Spencer, Robert S. Laramee, Guoning Chen, and Eugene Zhang. 2009. Evenly Spaced Streamlines for Surfaces: An Image-Based Approach. *Computer Graphics Forum* 28, 6 (2009), 1618–1631.

John C. Steuben, Athanasios P. Iliopoulos, and John G. Michopoulos. 2016. Implicit slicing for functionally tailored additive manufacturing. *Computer-Aided Design* 77 (2016), 107 – 119.

MFX Sylvain Lefebvre. 2017. IceSL, a slicing software. (2017).

Jun Wu, Charlie CL Wang, Xiaoting Zhang, and Rüdiger Westermann. 2016. Self-supporting rhombic infill structures for additive manufacturing. *Computer-Aided Design* 80 (2016), 32–42.

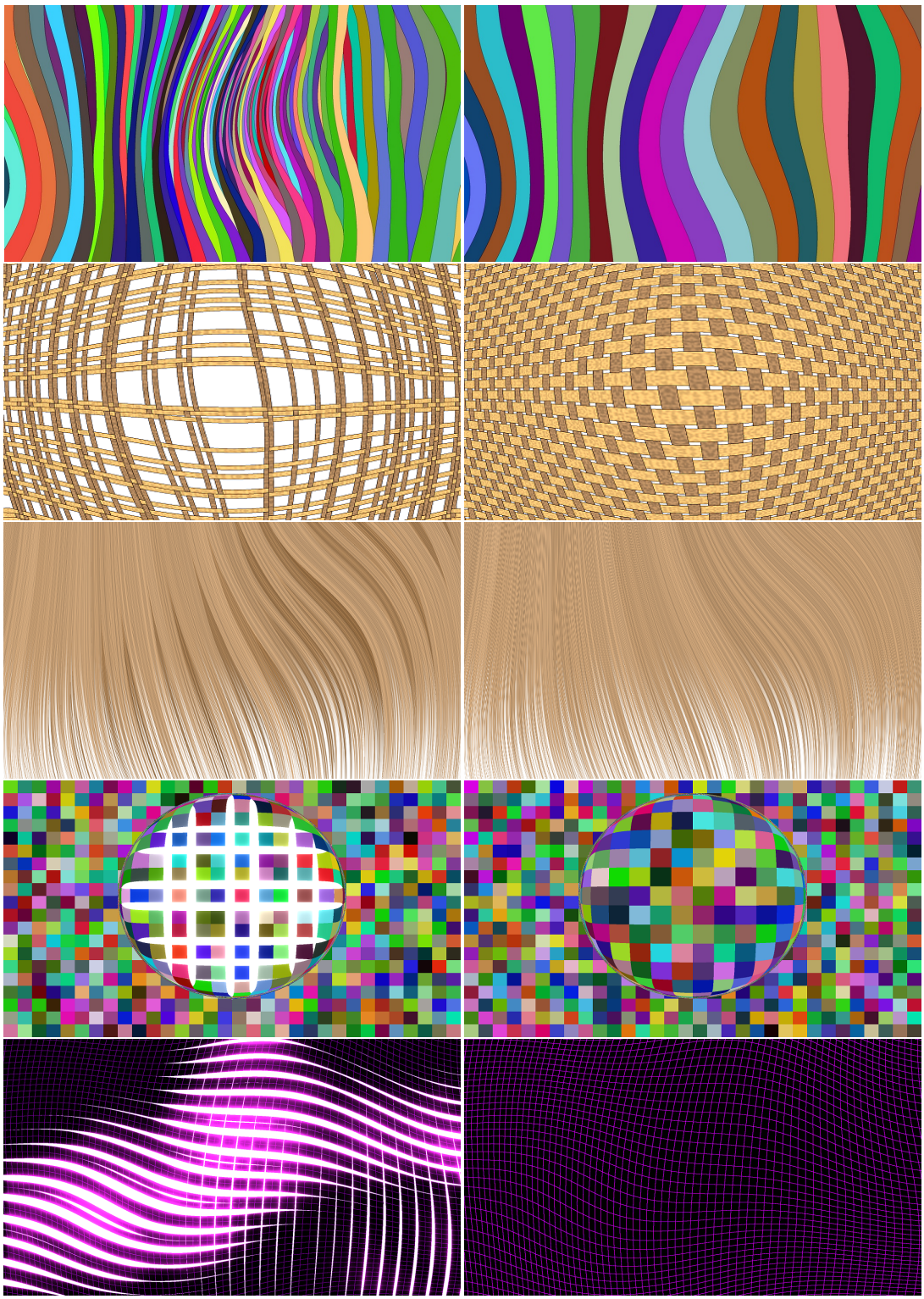


Figure 8: Various animated effects. Each pair is showing the effect with our technique with and without enabling the field d (without the bands do not adapt). Please refer to text for details and Table 1 for links to shadertoy implementations.