Discrete Texture Design Using a Programmable Approach

Hugo Loi, Thomas Hurtut, Romain Vergne, Joëlle Thollot

Discrete Texture Design Using a Programmable Approach

Hugo Loi[1]*, Thomas Hurtut[2], Romain Vergne[1], Joëlle Thollot[1]
[1]Inria-LJK (U. Grenoble, CNRS) [2]LIPADE - U. Paris Descartes

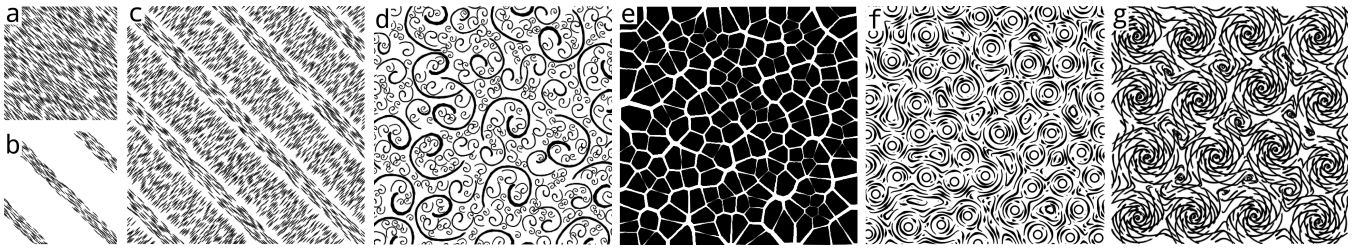

Figure 1: *In our approach each texture is the result of a simple program. Based on a set of combinable operators, a large variety of distributions can be intuitively described and produced. See the scripts corresponding to these images in supplemental material.*

1 Introduction

Many rendering methods use discrete textures (planar arrangements of vector elements) instead of classic bitmaps. Discrete textures are resolution-insensitive and easily allow to modify the elements' geometry or spatial distribution. However, manually drawing such textures is a time-consuming task. Automating this production is a long-time studied subject. The methods designed for this purpose deal with a difficult tradeoff between the reachable variety of textures and the usability for a community of users.

Numerous models have been proposed for representing discrete textures. These methods can be roughly classified into two categories: local models, that constrain the neighborhood structure of elements, and parametric models, that describe the global statistics of elements' spatial distribution. In order to be usable by artists, these approaches are mainly designed to allow by-example texture synthesis. However, the variety of textures they can achieve is limited by the difficulty for parametric models to represent local structures, or by the low ability for local models to capture large scale effects.

In this talk, we show that considering discrete textures as *programs* allow for a larger variety of textures than relying on a given model. These programs combine operators that distribute points, curves or regions on the plane. We propose a small set of such operators and we let the user write the texture program by combining them, such as in [Grabli et al. 2010] for stylized line drawing. The variety of textures reachable by this representation is then bounded by the combinatorial of the proposed operators. In return, we adress a different community of users, such as technical directors, due to the required use of a programmable interface. A larger audience can be attended by hiding the programming langage with a graph-based visual notation.

Contributions

- We propose a small set of atomic operators that distribute points, curves and regions on the plane, and we show their efficiency for representing a large variety of discrete textures.
- We factorize the redundant concepts between classic element distribution algorithms such as dart throwing and Lloyd's relaxation, by expressing them with our atomic operator set. This allows us to design a large number of variants of these algorithms that share the same factorized representation.

*This work has been sponsored by the ANR MAPSTYLE #12-CORD-0025 and ANR SPIRIT #11-JCJC-008-01.

2 Programs and Operators

We provide a set of operators for manipulating *scalars* (boolean, integer or real values) or *elements* (points, curves and regions in the 2D plane), and a system drawn from functionnal programming for combining these operators as functors. We propose a generic and factorized formulation of greedy (without backtrack) element distribution algorithms:

`while (`**`loop condition`**`) do {`
- **`pinning`**: create a new point p
- **`shaping`**(p): create a region r given p
- **`checking`**(r): decide to keep r or not}

Each of the four bold terms above denotes a user-chosen operator. Figure 1a shows a classic anisotropic dart throwing obtained with our generic algorithm when the user chooses a maximum iteration number as loop condition, a random point pinning operator, a hatch shaping operator and a non-overlap checking operator. Figure 1b shows a result with same choices but a checking operator that keeps regions contained in rectangles previously distributed regularly with the same algorithm. Figure 1c shows the combination of 1b and another distribution whose checking operator keeps the regions outside the rectangles. Figure 1d differs from Figure 1a only on its shaping operator, which combines a random rotation and a continuous shrink of a user-drawn spiral element.

We propose a similar formulation with shaping and pinning operators for region-based relaxation methods, including Lloyd's algorithm. Figure 1e shows a greedily-distributed set of regions, moved by our generic relaxation method and finally reshaped with a transformation of their Voronoï cells. Figure 1f shows a similarly distributed and relaxed set of circle regions. It is combined with the same algorithm than for 1a, but with a shaping operator that creates stylized stream lines around the distribution of circles. Figure 1g interlocks greedy distributions with pinning along curves and stream line shaping on the basis of a relaxed distribution of circles.

Our operator set is highly extensible and combinable, allowing a better adaptation to user's needs and further distribution concepts. See our supplementary document for our operators' reference and the programs corresponding to Figure 1.

References

GRABLI, S., TURQUIN, E., DURAND, F., AND SILLION, F. X. 2010. Programmable rendering of line drawing from 3D scenes. *ACM Trans. Graph. 29*, 2, 1–20.