gTangle: a Grammar for the Procedural Generation of Tangle Patterns

Christian Santoni Fabio Pellacini
Sapienza University of Rome

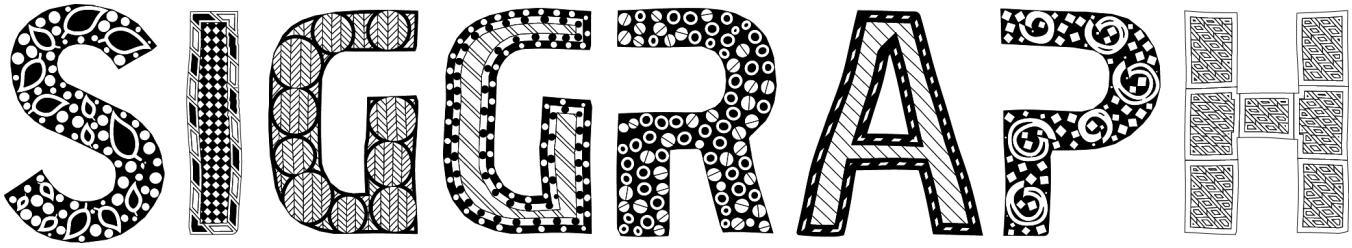


Figure 1: *An example* tangle *generated by our* group grammars*. Every letter is decorated by a different set of patterns, displaying the expressive power of our formal grammar. We generated this tangle by recursively combining, in a meaningful manner, our grouping, geometrical and decorative operators, all of which are well-defined on sets of arbitrary polygons with holes.*

Abstract

Tangles are a form of structured pen-and-ink 2D art characterized by repeating, recursive patterns. We present a method to procedurally generate tangle drawings, seen as recursively split sets of arbitrary 2D polygons with holes, with anisotropic and non-stationary features. We formally model tangles with group grammars, an extension of set grammars, that explicitly handles the grouping of shapes necessary to represent tangle repetitions. We introduce a small set of expressive geometric and grouping operators, showing that they can respectively express complex tangles patterns and sub-pattern distributions, with relatively simple grammars. We also show how users can control tangle generation in an interactive and intuitive way. Throughout the paper, we show how group grammars can, in few tens of seconds, produce a wide variety of patterns that would take artists hours of tedious and time-consuming work. We then validated both the quality of the generated tangles and the efficiency of the control provided to the users with a user study, run with both expert and non-expert users.

Keywords: tangles, procedural generation, grammars

Concepts: •**Computing methodologies** → **Shape modeling;** Texturing;

1 Introduction

Tangles are a form of abstract and structured 2D art [Roberts and Thomas 2012]. Tangles are black-and-white ink drawings created with the use of only a small set of basic strokes: dots, straight lines, simple curves and circles. Their complexity comes from the repetition of those strokes to form structured patterns, that often display clear recursive traits. Since tangles are drawn completely free-handed, without using any ruler or stencil, the structured patterns have an organic feel to them. Tangles are drawn recursively at different scales, starting from the bigger subdivisions, passing through the distribution of sub-structures over those areas, finishing with fine tangle patterns. An example of an hand-drawn tangle is provided in Fig. 2

Since their distinctive repetitive traits, the use of fine-grained features, and the high variation of patterns even in the same drawing, the creation process for a tangle can take up to hours, even for a skilled artist. Moreover, the completion of a non-trivial tangle, i.e.: which doesn't require only the use of a single pattern, represents a task with a steep learning curve for a non-expert user. These are the main reasons explaining why the existence of a tool that can automatically generate such patterns can aid artists in quickly testing ideas, prototyping concepts and produce final images that otherwise would have being too time consuming, or even impossible for non-experts.

The main goal of this paper is the procedural generation of tangles, considered as a special class of anisotropic non-stationary recursive patterns. We propose a formal model for tangles with enough expressivity to enclose both their high-level and structured features, together with their low-level details. We model tangles with a *group grammar*, a stochastic grammar that can be seen as an extension of *split grammars* [Stiny 1982; Wonka et al. 2003; Schwarz and Müller 2015], that specifically handles repeated and recursive patterns. We define a small set of basic operations that combined with the grammar can generate a large variety of patterns. Furthermore, we demonstrate how our model supports control over the generated tangles, allowing artists to finely customize their final results with a simple and efficient graphical interface. Finally, we run a user study to validate both the quality of the tangles generated automatically with group grammars, and the efficiency of the control provided to the users to customize generated tangles.

We implemented our grammar in a prototype system that can generate a large variety of tangle patterns, shown throughout the paper, automatically or interactively under user control. Fig. 1 shows an example tangle generated automatically by our system. We believe that the main contributions of our work are: (1) the formalization of tangles with a group grammar, (2) a compact set of operators that are efficient, closed under any set of arbitrary 2D polygons with holes, and that are able to encode a wide amount of variations in the generated tangles; (3) the design of some intuitive user interactions that allow user to both finely refine small-scale patterns and

SA '16 Technical Papers,, December 05 - 08, 2016, , Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2982417

ACM Reference Format
Santoni, C., Pellacini, F. 2016. gTangle: a Grammar for the Procedural Generation of Tangle Patterns. ACM Trans. Graph. 35, 6, Article 182 (November 2016), 11 pages. DOI = 10.1145/2980179.2982417 http://doi.acm.org/10.1145/2980179.2982417.

large-scale structures in the generated tangles.

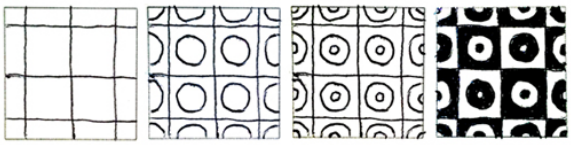


Figure 2: *A step-by-step example of an hand-drawn tangle, showing the grouping and recursive shape splitting nature that characterizes this form of art.*

2 State of the Art

Grammar Systems. Formal grammar systems have been presented over time for a variety of tasks related to content creation, and we consider them the most appropriate way to solve the problem of tangles procedural generation, since they can easily represent their recursive structure, can be extended to support users' edits, and follow a synthesis process that is more similar with respect to what artists actually do.

Over the years, L-systems have been proposed as tools for the procedural generation of plants [Prusinkiewicz and Lindenmayer 2012]. This approach has been greatly extended with the possibility to support computer simulation [Měch and Prusinkiewicz 1996], user generated inputs [Prusinkiewicz et al. 2001] and context sensitivity [Parish and Müller 2001]. In general though this kind of approaches follow a completely different generation process, which is more growth-like. This obviously is directly related to the way plants or trees are naturally structured, but doesn't fit well tangles, that are mostly generated by a process of recursive splitting.

Shape grammars are a formalism introduced in [Stiny 1980]: in their original formulation, this kind of grammars operated on labeled arrangements of points and straight lines. They showed to be a great tool in the analysis and design of architectural styles [Stiny 1980], but the model complexity made an usable computer implementation unfeasible, often forcing the derivation of such grammars by hand. Later, set grammars were presented [Stiny 1982; Wonka et al. 2003], as a simplification of shape grammars. Various extensions have been presented for such kind of grammars, like [Wonka et al. 2003] for architectural geometric details and [Li et al. 2011] for patterns on surfaces. [Schwarz and Müller 2015] further extended this concept augmenting the grammar model with first class citizenship for shapes, allowing for better context sensitivity and expressivity of the model itself.

Note that all the approaches presented in this section are modeled for specific kinds of content generation, e.g.: plants, building facades, cities, and to the best of our knowledge, our is the first work trying to deal with complex hand-drawn structured art.

Procedural Texture Generation. Procedural texture generation methods are a popular way to deal with the production of complex natural patterns [Ebert 2003]. Procedural generation is efficient and predictive, but requires different algorithms for different patterns. This paper presents a procedural method for generating tangles, a specific class of patterns that was not previously explored.

Example-Based Texture Generation. Example-based texturing overcomes the main limitation of procedural generation by producing results that are perceptually similar to a given texture exemplar, usually generating textures in larger domains with respect to the exemplar [Wei et al. 2009]. Most of these algorithms though, are based on sampled representations such as pixels [Efros and Leung 1999], vertices [Turk 2001] or voxels [Kopf et al. 2007]. Without any specific representation for discrete elements, arbitrary polygons in our case, synthesis becomes an arduous task for structured patterns, prone to artifacts or invalid results, e.g. open or overlapping shapes. We demonstrate this running a comparison with [Lefebvre and Hoppe 2005], which follows a similar approach to the works just described. Texture synthesis algorithms that have a concept of discrete elements have been indeed presented, such as [Ma et al. 2011] and [Ijiri et al. 2008]. Still, this kind of approaches are based on a neighborhood similarity concept, that in most cases doesn't apply to tangles. In fact, in most of the cases, even adjacent areas in a tangle are decorated with completely different techniques. Moreover, even if dealing with discrete elements, these approaches cannot represent the intrinsic recursive nature of tangles, thus limiting the range of achievable results.

Sketch synthesis. Most of the systems presented for the reproduction of handmade sketches aid the artists by handling repetitive tasks, such as structured drawings [Cheema et al. 2014], decorative patterns [Lu et al. 2014] and pen-and-ink illustrations [Kazi et al. 2012]. Note that all these systems need direct input by the users, while our main focus is on automatic generation. Another interesting approach is the one presented in works like [Xing et al. 2014] and [Xing et al. 2015], that describe systems where the artists are aided in the sketching process by an autocomplete feature, that tries to replicate users' edits to other parts of the image. Also this technique, using heuristics based on neighbourhood similarity, doesn't take into account the tangles recursive structure. Moreover, as also stated by the authors of the papers, this approach can fail with rapidly changing contexts, which breaks our desiderata to define operations that can be applied to any arrangement of arbitrary polygons with holes.

3 Tangle grammar

We represent a tangle as a set of adjacent 2D shapes constructed by recursively subdividing an initial configuration of shapes. In our implementation shapes are arbitrary 2D polygons with holes, where their boundaries are represented as highly-tessellated closed curves. We introduce group grammars, derived from set grammars, to automatically generate tangles from any valid starting arrangement of shapes. The core idea of our formalization is that all the grammar operators work on groups of shapes, both in input and output, subdividing the plane in new groups, or re-labeling the shapes in meaningful ways.

Tangle grammars, differently from other models derived from set grammars, explicitly express the grouping information of every shape, in a way that this information is directly accessible and modifiable by specific operators. More formally we uniquely identify a shape S as

$$S = \langle t, g, s; \Theta, \mathbf{d} \rangle$$

where: (1) t represents the type of a shape in the grammar context, and can be either a terminal or non-terminal symbol; (2) g is an unique identifier for the group to which the shape belongs to; (3) s is the shape id, that identifies the shape in its group, i.e.: when a new group is created, to each belonging shape is assigned an incrementing id s. These attributes are used for rule matching and can be modified by grouping operators (Section 4). In addition to that, each shape is described also by its additional attributes Θ, the list of curves describing the boundary of the polygon, and a unit vector $\mathbf{d}$, that defines the orientation of the shape. These attributes represent the data on which our geometric and decorative operators work on (Section 4). In the reminder of this section, we will omit Θ and $\mathbf{d}$ for readability.

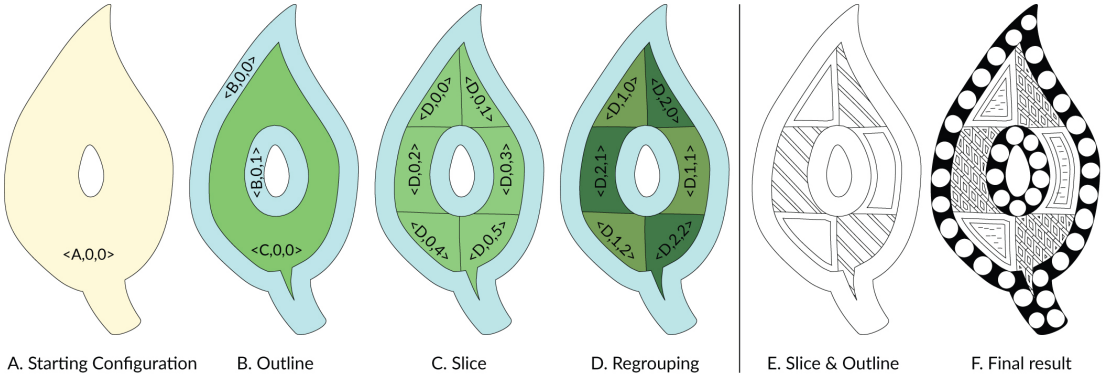


Figure 3: Left: *First steps in the expansion process of a toy group grammar. (A) The expansion starts from a set of arbitrary shapes, that in our system are represented as highly-tessellated polygons with holes. We identify shapes with $S = \langle t, g, s \rangle$, where t is the shape type, g identifies the shape group and s identifies the shape within the group. Here shapes are colored with a hue depending on t and a brightness depending on g. (B,C) The application of geometric operators subdivides the originating shapes in a set of new shapes, labeled accordingly to the rule semantic. (D) The application of grouping operators relabels shapes that belong to the same group, partitioning them in two or more new groups.* Right: *(E) The result of the expansion process after the application of a few more rules. Note how the grouping allows to obtain a checkerboard-like pattern. (F) Final generated tangle after all rules are applied.*

Once defined the representation of a shape, a tangle $\mathcal{T}$ can thus be written as a set of adjacent shapes S_i with

$$\mathcal{T} = \{\langle t_i, g_i, s_i \rangle\}_{i=1}^{n}$$

In our formulation, shapes with the same group id have the same type, but not viceversa. This allows us to express more elaborate grouping configurations, considering also shapes of the same type.

Starting from an initial set of shapes $\mathcal{T}_0$, productions are applied sequentially to the current state $\mathcal{T}_i$ of the tangle, to produce the next step $\mathcal{T}_{i+1}$ in the expansion process, until all shapes are terminals. An example tangle from a toy grammar is shown in Fig. 3. In group grammars, a production R is expressed in the following form

$$R = O(\{p_o\}) : t_m \rightarrow [\langle t_0, g_0 \rangle, \ldots, \langle t_k, g_k \rangle]$$

where O is the tangle operator that will be used when the rule is applied, $\{p_o\}$ the operator parameters, t_m is the tag that will be used to select the matching shapes $S_j = \langle t_m, g_m, s_j \rangle \in \mathcal{T}_k$, and $[\langle t_0, g_0 \rangle, \ldots, \langle t_k, g_k \rangle]$ represents the tags and group ids that will be assigned to subsets of the shapes resulting from the subdivision produced by the rule application, accordingly to the semantics of the operator (see Section 4). Group ids are generated incrementally starting from the largest one in $\mathcal{T}_i$, while shape ids are generated incrementally within each group.

A single step of the expansion process, executed on $\mathcal{T}_i$ to produce the next tangle arrangement $\mathcal{T}_{i+1}$, is comprised of the following steps: (1) we first select the set of non-terminal shapes $\mathcal{S}$ with lowest group id in $\mathcal{T}_i$, i.e. $\mathcal{S} = \{\langle t, g, s_j \rangle \in T_i\}$ such that $g = \min(g \in \mathcal{T}_i)$, thus executing a breadth-first expansion in our grammar; (2) we then find the set of rules $\mathcal{R}_m$ with the same t_m as the tag of given shapes, i.e. $\mathcal{R} = \{R : R.t_m = t\}$; (3) from these, we choose randomly a rule R from $\mathcal{R}$. (4) we generate a new set of shapes $\mathcal{S}_N$ by applying the operator referred by rule R to each shape S_j in $\mathcal{S}$, where $\mathcal{S}_N = \{S_k : S_k = \langle t'_k, g'_k, s'_k \rangle$ with $t'_k \in \{t_0, \ldots, t_k\}\}$; (5) we finally update the tangle $\mathcal{T}_{i+1}$ as $\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{S} + \mathcal{S}_N$ where $+$ and $-$ are respectively set union and set difference.

4 Operators

The operators applicable during the expansion process are subdivided into three main categories: (1) *grouping operators*, that don't actually modify any shape, but only re-label sets of shapes by manipulating their group and shape ids, (2) *geometric operators*, that subdivide shape groups by splitting, outlining and placing objects in them, and (3) *decorative operators* that simply modify the final appearance of a shape, without further subdivisions. We support two decorative operators: *filling* that sets the background color of a shape and *stippling* [Secord 2002] that stipples the shape with geometrical details like dots, dashes, curves.

4.1 Grouping operators

Observing how tangles are made by artists, it's easy to see how they resort to different kinds of meaningful grouping between the drawn shapes, rearranging them in a meaningful manner, that can also be independent from the step at which such shapes were created (see for example Fig. 2, where shapes drawn at different steps are then colored following an alternating pattern).

To formalize this process, we introduce grouping operators in our *group grammars*. Note also that by supporting explicitly groups of shapes with group ids, the grammar complexity is drastically reduced, since the same rules, after a re-arrangement of the groups, can be used again in a entirely recursive way, in the same way tangle art is intended. In Fig. 4 we show the results of the application of different grouping operators on the same expanded tangle, to highlight some of the different obtainable variations.

We support an *ungroup* operator that takes all shapes in $\mathcal{S}$ and assigns a new group id g to each one as

$$ungroup() : t_m \rightarrow \left[\langle t', g_0 \rangle, \langle t', g_1 \rangle \ldots \langle t', g_k \rangle\right]$$

We also support a *regroup* operator. This operator rearranges the shapes of a selected group into multiple newly created groups, assigning them accordingly to the operator parameter k. When applying this operator, a list $G = [g_0, \ldots, g_k]$ of new group ids is generated and to each shape $S_j = \langle t_j, g_j, s_j \rangle \in \mathcal{S}$ is assigned a new group id $g_j = s_j mod\, k$.

$$regroup(k) : t_m \rightarrow \left[\langle t', g_0 \rangle \ldots \langle t', g_0 \rangle \ldots \langle t', g_k \rangle \ldots \langle t', g_k \rangle\right]$$

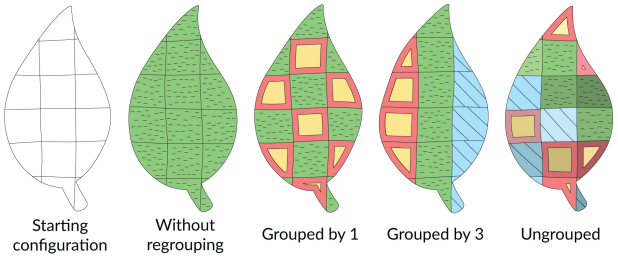


Figure 4: *Different grouping operators applied on the same starting tangle. Grouping rules control sub-pattern distributions in a manner that is common in tangle art.* From left to right*: Starting tangle, no grouping applied, chessboard pattern [regroup(1)], column pattern [regroup(3)], random pattern [ungroup()]*

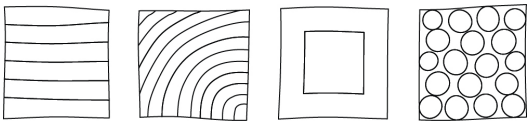


Figure 5: *Geometric operators applied to a square.* From left to right*: regular split, streamline split, outline and shape placement.*

4.2 Geometric operators

Geometric operators subdivide shapes into more detailed ones. Fig. 5 shows the trivial application each one of them, while Fig. 6 and Fig. 7 show some examples on how they can be combined together to obtain more complex patterns.

Regular Split. The regular split operator takes as parameters an arbitrary curve c, a list of offsets o and a flag x. Then, it subdivides every shape S in $\mathcal{S}$ with boolean operations, placing copies of c along the shape orientation $\mathbf{d}$. The copies have fixed distance if only one offset if specified, or have variable distances if multiple are provided (i.e.: the current offset for a newly added curve is chosen cycling through the offsets list). The parameter x decides if the split is dual or not: in a dual split, the splitting process is done an additional time, orthogonally to the shape orientation $\mathbf{d}$, to obtain a grid pattern.

$$regularSplit(c, o, x) : t_m \rightarrow [\langle t_0, g_0 \rangle, \ldots, \langle t_0, g_0 \rangle]$$

Geometric Outline. The outline operator splits every shape in $\mathcal{S}$ accordingly to their geometric outline, shrinking the shape borders by a fixed distance d. We compute this with a standard polygon outline algorithm. This operator produces an outer remainder and a set of inner shapes and can assign two different tags for each category:

$$outline(d) : t_m \rightarrow [\langle t_{out}, g_0 \rangle, \langle t_{in}, g_1 \rangle, \ldots, \langle t_{in}, g_1 \rangle]$$

Streamline Split. This operator is used to split every shape in $\mathcal{S}$ in different and more complex ways that are not achievable with regular splits. The parameter $type$ defines what kind of split the operator will perform on the matching shapes, while o is the split offset:

$$streamlineSplit(type, o) : t_m \rightarrow [\langle t_0, g_0 \rangle, \ldots, \langle t_0, g_0 \rangle]$$

We implemented a method that is based on drawing complete, equally-spaced streamlines guided by vector fields, designed for different effects. A streamline can be considered as the path traced by an imaginary massless particle placed into a fluid flow, described by a vector field. The tracing of this path consists of solving a differential equation, that in our case is approximated with Euler integration. In order to draw the streamlines, we compute the vector field

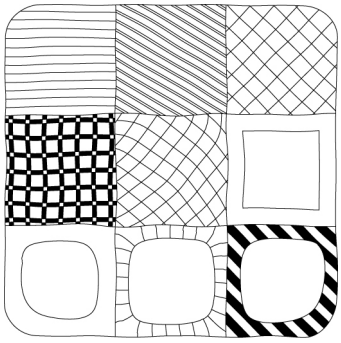


Figure 6: *Complex patterns generated by geometric operators with different parameters.* Top, from left to right*: Regular split, regular split with multiple offsets, dual split.* Middle, from left to right*: Dual split with multiple offset, streamline dual split, geometric outline.* Bottom, from left to right*: smooth outline, smooth outline with circular split, smooth outline and regular split.*

inside the shape, by setting values on its boundaries and then interpolating those values using generalized barycentric coordinates [Meyer et al. 2002].

For a particular point P inside a shape S, the associated field vector $\mathbf{v}$ is computed following this steps: (1) a vector $\mathbf{v_i}$ is assigned to every point $P_i \in [P_1, \ldots, P_k]$ of the shape boundaries; (2) the barycentric coordinates $(w_1, \ldots, w_k)$ for P are computed using the method presented in [Meyer et al. 2002]; (3) the final field vector $\mathbf{v}$ is computed, interpolating the boundary vectors with the barycentric weights: $\mathbf{v} = \sum_{i=1}^{k} w_i * \mathbf{v_i}$.

We set the boundary vectors for P_i differently to achieve different splits. In our prototype we support: (a) *smooth outline:* $\mathbf{v_i} = P_{i+1} - P_i$; (b) *strip split:* $\mathbf{v_i} = (P_{i+1} - P_i) + (P_{i-1} - P_i)$ (if $\mathbf{v_i}$ points outside the shape, it's inverted); (c) *circular split from a point* P'*:* $\mathbf{v_i} = [(P' - P_i)/|P' - P_i|]^{\perp}$. Note that, as a final step, all vectors $\mathbf{v_i}$ are normalized. Examples of such effects are shown in Fig. 5 and Fig. 6.

Shape placement. The placement operator distributes an arbitrary shape S' in all shapes $S \in \mathcal{S}$. With this operator several types of packing can be produced, as shown in Fig. 7. The operator attributes are the size of the placed shapes s and their minimum desired distance d.

$$place(s, d) : t_m \rightarrow [\langle t_{rem}, g_{rem} \rangle, \langle t_0, g_0 \rangle, \ldots, \langle t_0, g_0 \rangle]$$

Soft packing of arbitrary shapes is a well-known and still open problem, so we provide an approximate solution for it, to knowing that other solutions might work as well, like, for example, following a similar approach to [Kindlmann and Westin 2006]. The procedure goes as this: (1) firstly, we estimate the centers of the shapes to be placed with a Poisson distribution, computed with dart throwing; (2) we then optimize the centers position computing a Centroidal Voronoi diagram, using Lloyd's relaxation. Step 1 allows for a faster convergence for the relaxation step. Since simple Centroidal Voronoi diagrams are well-defined (i.e.: that would produce a set of points all inside the voronoi cells, which are the only ones valid for placement) only for convex regions, we use Geodesic Centroidal Voronoi diagrams. For this technique, we implemented an approximate approach that computes a discrete solution, using Dijkstra algorithm, on a raster grid with a resolution higher than the polyline resolution, and then snaps the centers of each cell to the closest point on its boundary, since cell centers might still lie outside its boundary; (3) once the centers are computed, we place

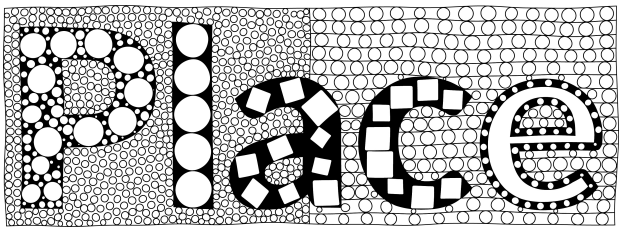


Figure 7: *Examples of the place operator. Note how our method can produce both organic and regular patterns. The latter are obtained by applying the place operator after an outline or a split.*

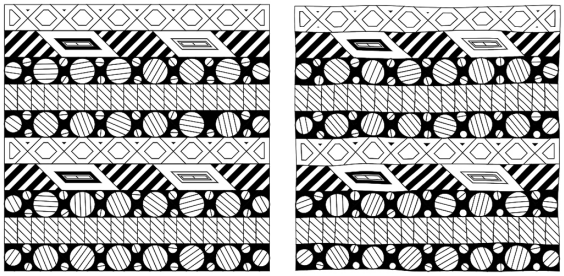


Figure 8: *Example of a tangle generated with (*right*) and without (*left*) shape perturbation. Note how perturbation makes the tangle feel more hand-drawn.*

the objects in the shape; (4) finally, in the case that there are still overlapping objects, we perform a final step in which we slightly deform the placed shapes, scaling them down until no overlapping objects are present.

4.3 Shape perturbation

Tangles are drawn completely free-handed. To simulate the organic feel of our generated tangles, we apply a perturbation for all subdivided shapes. Since we are dealing with adjacent close polylines, meaningful intersection points and shape adjacencies might be invalidated if perturbation is done improperly. For this reason, we use a *coherent noise function* that is applied to all shapes with the same group id, following an approach similar to [Zainab 2009] and applying a perturbation based on Perlin noise, seeded by the group id. Thus, each point in the shape polyline is perturbed by an amount dependent on the noise function, defined by its amplitude and frequency, that at the moment are designed as constants in our system. In Fig. 8 we show how this process increases the results' fidelity to hand-drawn tangles.

5 Results and discussion

All the tangles presented in this paper were generated by different grammars, except for the last one (Fig. 18) which was created by aggregating all the grammars used for the other examples, and then further extended with additional productions. Tab. 1 summarizes statistics on the presented grammars. Throughout this paper we already shown many tangles generated by our grammars. The choice of the presented operators was tailored around the creative process and fundamental characteristics of tangles: when combined, together with the hierarchical nature of group grammars, they can produce a wide amount of variations, as shown in Fig. 10. We now show three additional results: in Fig. 17 we show how users can finely control the expansion process of a tangle, modifying the final result to better adapt it to their likings; the images in Fig. 11 were instead generated without any user intervention, and displays the results generated from two different grammars, one designed with a stronger focus on splitting and regrouping operations, the other on placement of objects of different nature.

Name	Nodes	Edges	Depth	Steps	Shapes	Timing	Design time
siggraph	48	65	9	216	1624	1.6s	15m
tree	22	29	8	42	3096	2.6s	15m
albert	25	35	7	85	1590	1.43s	20m
tiger	33	44	12	201	3650	2.84s	30m
village	101	137	17	907	18396	190.5s	(*)

Table 1: *This table resumes the information about the complexity of the grammars and the tangles, showed in the paper, that were generated by them. (*) No design time is showed for* village *since it was created by simply aggregating the grammars already presented.*

As an example, in Fig. 9 we show the structure of the group grammar used to produce the leaf in Fig. 3. The group grammars used to produce all the other results presented in the paper will be provided as supplemental material.

To better understand grammars' complexity, we define a graph representation for them: a *group grammar graph* is a directed graph $G = (V, E)$, where each rule R_i is represented by a node $v_i \in V$ and an edge (v_i, v_j) exists if the matching tag t_m of R_j is contained in one of the tags $[t_0, \ldots, t_k]$ of R_i. The graphs relative to the presented grammars ranged from 22 to 101 nodes, 29 to 137 edges, and extended between 7 and 17 levels of depth. The design of each grammar, written in text form similarly to the ones provided in supplemental material, took between 15 (*tree*) and 30 minutes (*tiger*). This demonstrates how all the grammars created are quite compact, but complex enough to correctly enclose the concept of recursive splitting that is at the base of tangles creation.

From the point of view of the generated tangles, the expansion process took roughly between 1.43 to 190.5 seconds, encompassing between 42 to 907 expansion steps. Results were generated on a 3.1Ghz machine. The final images extended from 7 to 22 inches in dimensions (diagonal of the bounding box of the starting image) and contained between 1590 and 18396 final shapes, drawn at a polyline sample resolution of 2 units. This, in correlation with the creation timings, shows how the system is stable, scalable and that produces results that, if done by hand, would have represented far more time consuming and challenging tasks, if not unfeasible at all.

1. $outline(25) : big\_poly \rightarrow [\langle in \rangle, \langle out \rangle]$
2. $place(20, 25) : out \rightarrow [\langle blob \rangle, \langle rem \rangle]$
3. $regularSplit(line, 90, 1) : in \rightarrow [\langle quads \rangle]$
4. $regroup(2) : quads \rightarrow [\langle ung\_quads \rangle]$
5. $outline(7) : ung\_quads \rightarrow [\langle in\_2 \rangle, \langle out\_2 \rangle]$
6. $outline(5) : in\_2 \rightarrow [\langle in\_3 \rangle, \langle out\_3 \rangle]$
7. $stipling() : in\_3 \rightarrow [\langle term \rangle, \langle term \rangle]$
8. $regularSplit(line, 50, 0) : ung\_quads \rightarrow [\langle slices \rangle]$
9. $regroup(2) : slices \rightarrow [\langle reg\_slices \rangle]$
10. $regularSplit(line, 10, 0) : reg\_slices \rightarrow [\langle final \rangle]$
11. $outline(3) : final \rightarrow [\langle tiny \rangle, \langle tiny\_out \rangle]$
12. $invert() : rem \rightarrow [\langle term \rangle, \langle term \rangle]$

Figure 9: *Group grammar used to generate the tangle showed in Fig. 3. For better readability, the rule formalism is reduced to* $O(\{p_o\}) : t_m \rightarrow [\langle t_0 \rangle, \ldots, \langle t_k \rangle]$

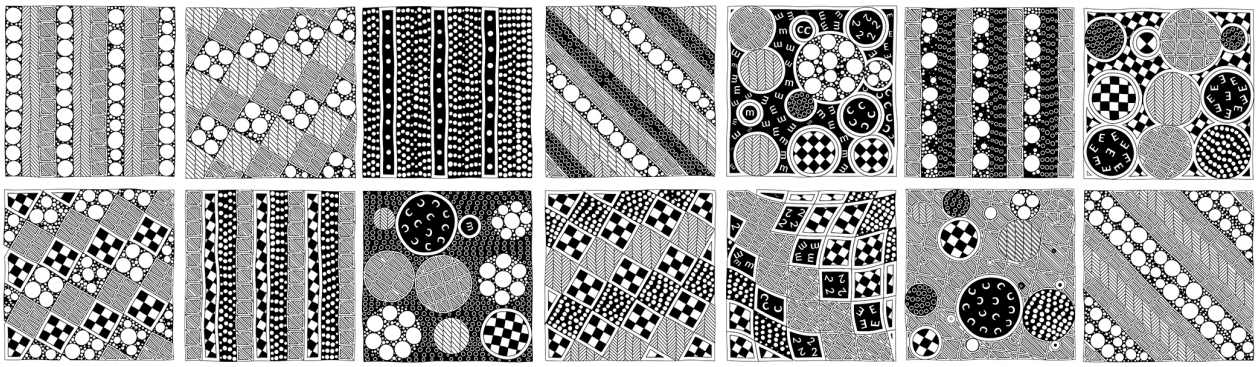


Figure 10: *In this figure we show tangles generated automatically, from a single grammar. Note how, even without user intervention, the results show a great amount of variation.*

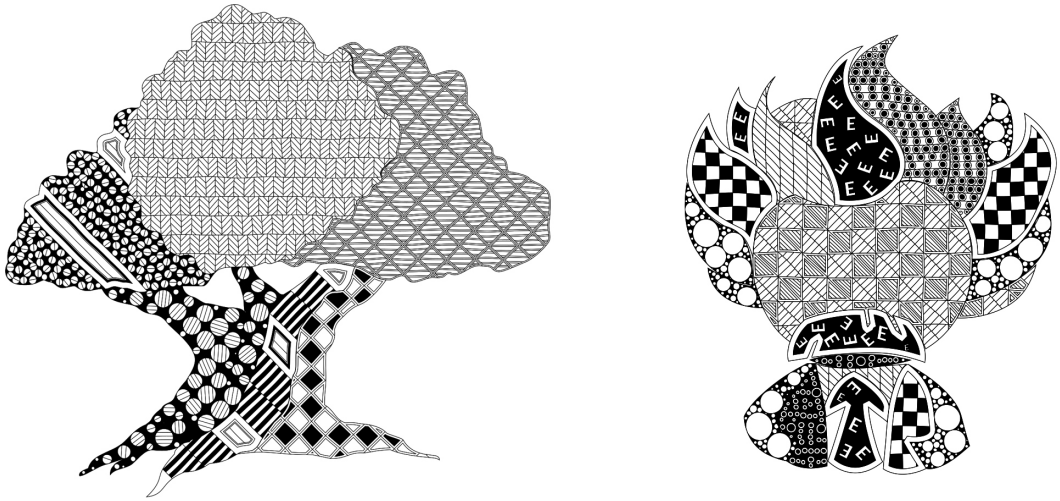


Figure 11: *Example tangles generated by two group grammars, without user intervention, in less than 8 seconds with on average 1400 drawn shapes.* Left: *Grammar with emphasis on split and grouping operators.* Right: *Grammar with emphasis on placements.*

5.1 User study - Classification task

We validated with a user study if the produced images could be recognized as tangles by users. We report this part here and send reader to section 7 to read the full experiment. In this task, participants were to match 6 images with their corresponding category, which were: textures, geometrical patterns, tangles, hand-drawn pictures. To be as fair as possible, each "non-tangle" category also showed some repetitive, hierarchical or artistically affine traits. This task was designed to evaluate the expressiveness of group grammars, and their ability to replicate tangles. The "not tangles" images (3 out of the total 6) has been placed to act as distractors. The classification measured the number of generated tangles correctly recognized as hand-made tangles and we validated the results running an exact binomial test. The 92.67% of the generated tangles were correctly classified and we rejected the null hypothesis with a p-value $p < 10^{-8}$.

5.2 Comparison with prior work

The geometrical and repetitive nature of tangles might suggest that the problem of their automatic generation could be solved with texture synthesis. We think that such a choice would end up in a more complex system, not able to produce results comparable to the ones group grammars are able to create. To support our thesis, we performed a comparison with some of the previous works that can be found in the vast literature produced in this subfield.

There are three main reasons that explain why we took a different approach for the generation of tangles. Firstly, most of the texture synthesis approaches are based on neighborhood similarity metrics or on patch stitching, that we think are not expressive enough to be able to encode the hierarchical structure of tangles, where even adjacent areas can display entirely different patterns. Secondly, most of the techniques developed are example-driven. This basically disables the possibility for the user to have a fine-grained control over the final result, even hindering the possibility for non-experts to produce their own tangles, without resorting to get already made tangles as exemplars. Finally, we think that to encode such a wide amount of variations in a single synthesis solution would produce an highly complicated system, whereas group grammars have shown to be a very compact model able to maintain an high level of expressivity.

Comparison with [Lefebvre and Hoppe 2005]. The first comparison was performed with [Lefebvre and Hoppe 2005], a texture synthesis scheme based on neighborhood matching, since it showed great results in generating textures with visible structured repetitions. The comparison was performed producing tangles con-

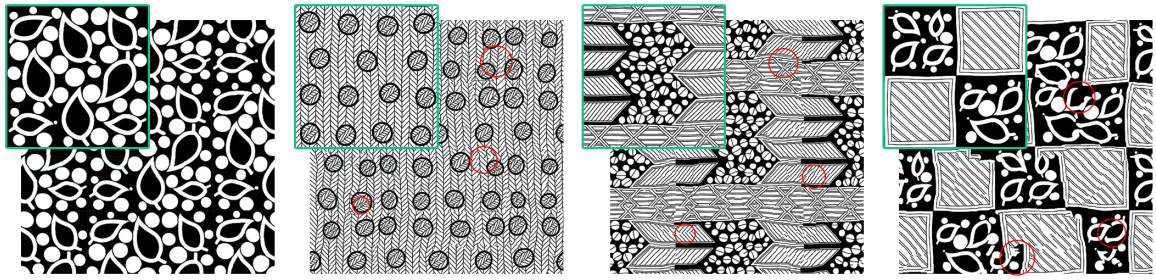


Figure 12: *The results of the comparison run with [Lefebvre and Hoppe 2005]. In the insets, the tangles produced by our system, that were used as input for the texture synthesis process. The algorithm performs well when the tangles were generated mostly from object placements, but fails to maintain a coherent structure when splits are used. In the red circles we highlight some of the inconsistencies (i.e.: open shapes, overlapping areas)*

strained to be in a square area, designing grammars that would produce images as close as possible to toroidal textures. The produced tangles were then used as exemplars for the system. As the results show (Fig. 12), the algorithm presented in [Lefebvre and Hoppe 2005] fails to correctly encode patterns that show the classical high-level structured features together with fine repetitive details. We think that this is directly entailed by the fact that a method based on computing similarities of "flat" neighborhoods cannot have enough expressive power to correctly represent recursive and hierarchical structures. Consequently, we think that similar approaches based on neighborhood similarity metrics, even when applied on the concept of discrete objects placement ([Ma et al. 2011]) would display the same artifacts in the final results.

Comparison with [Loi et al. 2013]. We then performed a comparison with the method proposed by [Loi et al. 2013], which casts the problem of generating discrete textures to a programming approach, akin to shading languages. [Loi et al. 2013] provides a set of atomic operators that distributes graphic elements (points, curves and regions) on a plane, together with operators that implements different aspect of classic element distribution algorithms.

Since this is a fully developed programming language, we could not re-implement it fully, so we performed a qualitative comparison between the two systems. We provided the authors of the paper with tangles automatically generated from our system, and then asked them if, for each tangle: (a) their system could reproduce such patterns; (b) how complex the program would have been to accurately reproduce the same image. All the results we show in this section were provided directly by the first author of the paper. To see the tangles that were provided for the comparison, and to read a more detailed analysis of all the steps required by [Loi et al. 2013] to reproduce each tangle, please refer to the supplemental material.

The author stated that the system could reproduce all the tangles we provided. To compare two generative systems as ours and [Loi et al. 2013], we chose to evaluate the complexity of the solution required to generate each tangle included in the comparison. Thus, we compared our group grammars complexity with the complexity of the programs needed to be written with the approach implemented by [Loi et al. 2013], counting respectively the number of rules in the grammar and the lines of code in the programs.

As showed by Table 2, group grammars demonstrate to be a more compact system with respect to [Loi et al. 2013]. Even with increasingly more complex generated tangles, grammars complexity remains stable, as the elaborateness of the final results emerges from the intrinsic recursive nature of the model itself. Moreover, since the system in [Loi et al. 2013] aims only to distribute elements, rather than procedurally them, the variations that are obtainable from the same program are more limited with respect to a group grammar.

Name	Grammar rules	Lines of code
tangle01	4	22
tangle02	22	55
tangle03	25	65
tangle04	33	75

Table 2: *This table shows the results of the qualitative comparison run with the system implemented in [Loi et al. 2013].* Grammar rules *and* Lines of code *represent, respectively, group grammars length and scripts length for [Loi et al. 2013] necessary to reproduce the comparison images.*

5.3 Limitations and future work

2D domain. As of now, the grammar is designed to act on arbitrary 2D polygons. If applied on 3D surfaces directly using a parametrization, the tangle would suffer from distortions and discontinuities that would break its patterns. We leave as future work the design of a grammar, and related operators, that generate tangles directly on 3D surfaces.

Grammar learning. All the grammars in the papers have been manually written, and at the moment there is no way to generate them automatically, for example from an image of a finished tangle. This makes exactly replicating a tangle time consuming. An interesting consideration though, is that often tangles are provided with step-by-step tutorials that describe the actual creation of the final result. These guides could be used to learn the structure of a grammar that could generate a similar result, both identifying the operators used at each step, and estimating their parameters.

New operators. Our grammar does not include operators typically found in set grammars, such as linear transformations. We don't have included them since they do not have an obvious counterpart in tangle generation, so we did not find the need to use them. The question remains whether they would significantly decrease the grammar complexity if they were available. We leave this tests as future work.

6 Extension: User control

We implemented our tangle grammar in a interactive system that is able to generate tangles automatically, starting from any initial set of shapes. This tool can be used both for the solely automatic generation of a tangle, following the expansion process described

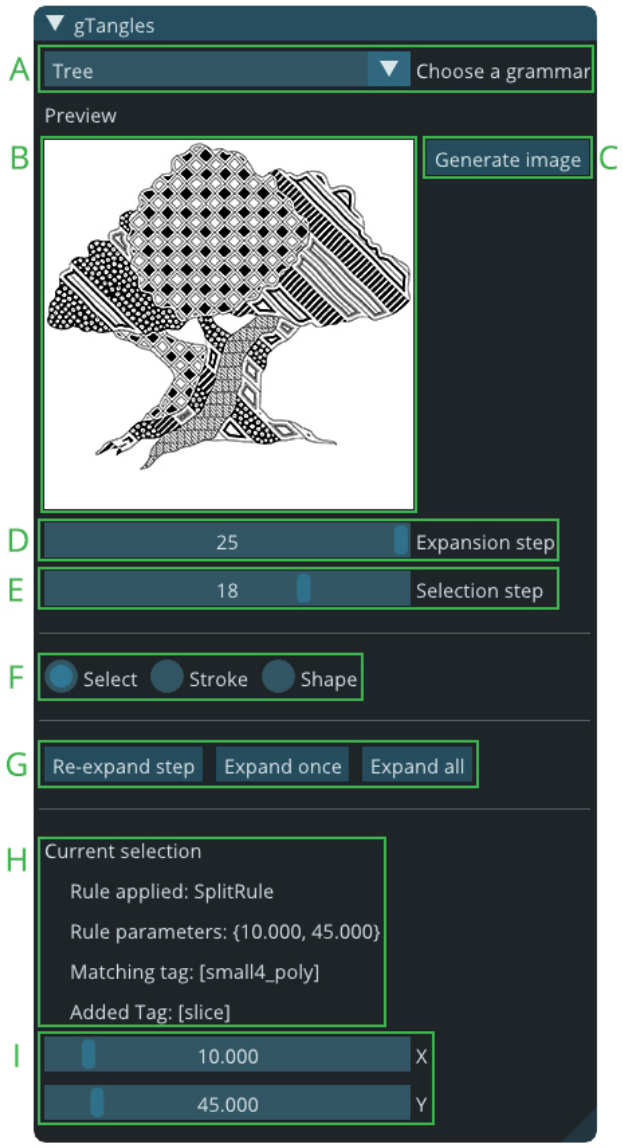


Figure 13: *Description of the system UI.* ***A****: dropdown list for grammar selection;* ***B****: preview of the tangle style achievable by the selected grammar;* ***C****: button used for starting the generation of a tangle;* ***D****: slider showing the current expansion step, that can also be used to visualize previous steps in the generation history;* ***E****: if a shape is selected, this slider shows the step in which such shape was generated;* ***F****: buttons used to switch between the* selection mode *and the* drawing mode*, used to provide new curves for the re-execution of a rule;* ***G****: button used for the re-expansion;* ***H****: if a shape is selected, this area shows data relative to the rule that created such shape;* ***I****: sliders used for parameters modification.*

in Section 3, and for the editing of an already generated tangle, to better adapt the final result to the user's likings.

To support user control over generated tangles, we provide three different interactions, already all available in the system, as showed in the supplemental video: (1) *history navigation*; (2) *rule re-expansion*; (3) *parameters modification*. All the interactions, together with some basic controls, are exposed in the interface showed in the supplemental video, as described in Fig. 13.

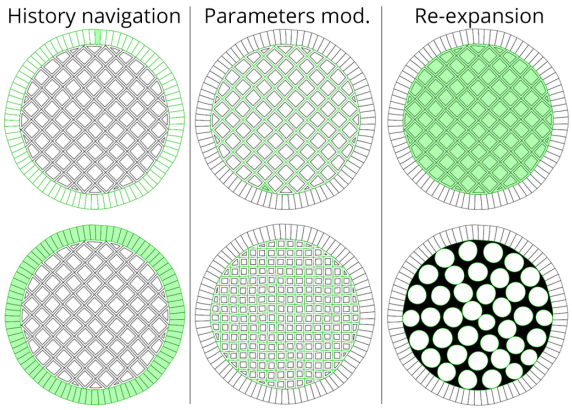


Figure 14: *Here we show the results of three possible interactions using the features introduced in Section 6. Top: selected shapes. Bottom: result after completing the interaction.*

History navigation. The interface allows user to move through the expansion history of a tangle, simply selecting a shape. Once a shape in the tangle is clicked, the system retrieves the expansion step that generated the shape group that contains the selected shape. Every additional click on the same shape retrieves the expansion step that created the set of matching shapes for the currently selected step. An example of this interaction is showed in Fig. 14. This allows users to easily modify any shape of a tangle, independently from the expansion step at which that shape was created.

Re-expansion. The system supports the re-execution of a specific expansion step. This allows for large scale edits, giving the user control over the distribution of patterns in the final tangle, as shown in Fig. 14. Once the user selects an expansion step and asks for a re-expansion, the system performs a series of tasks: (1) it selects all the steps that are dependent from the selected one, i.e.: whom matching set contained at least a shape added by the currently selected expansion or by another step already in dependency; (2) removes from the current tangle all the shapes added by such steps; (3) restarts the expansion process over the current tangle, filling the blank areas with new shapes, as described in Section 3.

Parameters modification. When an expansion step is selected, using history navigation, the user can interactively modify the parameters of the operator executed at that step, as shown in Fig. 14. This allows for fine refinements of the generated tangle, since the only shapes that are modified are the ones that were initially selected during the matching phase of the selected expansion, leaving the remaining tangle untouched. As showed in the supplemental video, all the described operators parameters are directly modifiable. For operators with input strokes or shapes, e.g.: the curve for a *regularSplit* or the polygon for a *place*, the user can provide them simply drawing them. For a detailed description of all the possible modifications, please refer to the supplemental video.

In figure Fig. 17, we show an example of how users can interactively modify an already generated tangle, result of a completed expansion process. In the supplemental video, we show a screen capture of the editing session.

7 User Study

We run a targeted user study to validate the tool and the generated tangles. We asked 16 subjects, both novices and experts in digital painting, to use the tool after a short training. We asked them to

complete specific tasks and we collected their feedback and opinions with a questionnaire.

Goal. The main goals of the experiment were the following: verify if the automatically produced images can be recognized as tangles; if novice and experts can have control on grammars expansions in matching a goal or just in complying their wills in fulfilling artistic taste; which kind of benefits the use of the tool can provide.

Experimental procedure. We collected 16 subjects between academia and artists. All of them had at least a minimum knowledge in the use of digital editing tools. We trained the subjects by presenting the tool features in a 5 minutes explanation where we showed the tool controls and interactions. For validating the comprehension of the interaction patterns, we asked the participants to complete the training performing some easy tasks (increasing an outline rule parameter, changing the shape of a place rule). We then provided a simplified interface asking users to perform the tasks listed below. At the end of the executive phase, we administered a questionnaire for collecting their feedback and opinions. The questionnaire contained also the classification task mentioned in Section 5.1. The whole experiment can be found in the supplemental material. We run experiments on users individually, and we provided to users full anonymity in the survey completion phase. None of the subject was paid to participate to the study. To avoid good subject effect on users who may already know about tangles, we never mentioned the specific word till the very end of the questionnaire.

Tasks. In targeting tasks T_{m1} and T_{m2} we asked subjects to start from an existing tangle and to modify it to match a target one. In Fig. 16 we report the starting and target images for both tasks. The maximum amount of time for task completion was 8 minutes. In task T_{exp} we asked expert users to reproduce the edits to match the target tangle of T_{m1}, using a digital editing tool of their own choice. The maximum amount of time for task completion was 30 minutes and we allowed users to skip it. The last task, T_{open}, was an open task on a complex tangle similar to Fig. 17, where users can edit the tangle to match their own taste, adjusting expansions parameters.

In the questionnaire we collected users feedback about complexity to match the target tangles, system control, the experience they had with the interface responsiveness and if the tool gave them some advantages compared to the use a usual digital editing tool for producing tangles. We also asked user to classify a set of provided pictures among manual drawings, geometrical patterns, natural textures and tangles.

User Study Results. Here we report the study results on the different tasks and on the questionnaire. All the users completed tasks T_{m1} and T_{m2} with an average time of 3.30 minutes on the 8 given. None of them skipped the task or withdrawed the completion. All expert users agreed that on task T_{exp} they could achieve the same matching task with other tools but all of them agreed that this would take a remarkable larger amount of time. Over the 7 experts, 5 refused to complete the task T_{exp}. We report some significant results of the run user study in Fig. 15. This lead us to conclude that the majors benefits in the use of the tool are the following: giving novices capability to produce tangles; more accuracy during the editing and time saving compared to already existing commercial tools; good control of the grammar expansions and operators parameters. The 100% of the subjects would recommend the use of this tool for the creation of digital tangles. We quote here some informal feedback we collected: *"...after the training the tool was really easy to use, also for people like me, beginners in graphics...regular spaces subdivision and outlines were frustrating and nonetheless imprecise in Adobe Illustrator. They took a lot of effort. In your tool those were immediate..."* .

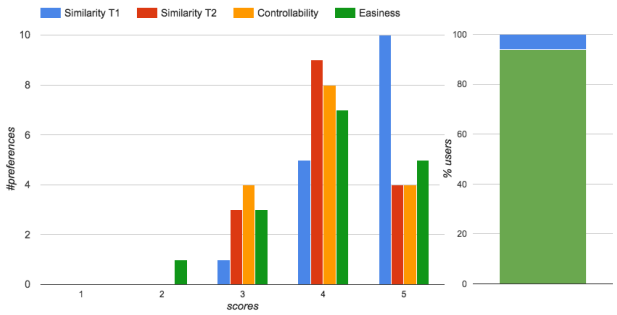
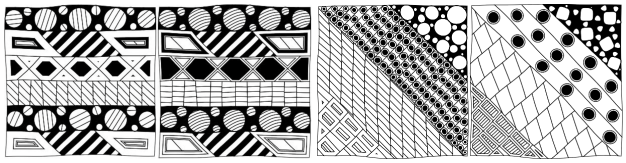


Figure 15: *User study results obtained in terms of controllability.* ***Left:*** *Users' rating of: similarity of the created tangle with T_{m1} and T_{m2} targets; system controllability and easiness to complete the task.* ***Right:*** *user perception of the control system. The 93.8% of the users found the system satisfying in the control it gave to them.*



(a) T_{m1} *begin* **(b)** T_{m1} *target* **(c)** T_{m2} *begin* **(d)** T_{m2} *target*

Figure 16: *Matching task performed in the user study: users have to adjust grammar parameters to achieve the target tangles. The matching could be achieved by different edits sequences.*

8 Conclusions

We presented an extension of set grammars, for the procedural generation of tangles. Our group grammar, and related operators, successfully capture the repetitive recursive patterns that make tangles artistically interesting. We plan to extend our approach to generate tangles on surfaces and learning tangles from examples.

9 Acknowledgments

We would like to thank Francesca Petetti for the SVG input images, Valentina Tibaldo for the user study, and Lynn Allen, Terry Lynn and Persephone Pomegranate for the handmade tangles shown in the video. This work was partially supported by MIUR, Sapienza University of Rome and Intel Corporation.

References

CHEEMA, S., BUCHANAN, S., GULWANI, S., AND LAVIOLA, JR., J. J. 2014. A practical framework for constructing structured drawings. In *ACM IUI '14*, 311–316.

EBERT, D. 2003. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann.

EFROS, A., AND LEUNG, T. 1999. Texture synthesis by non-parametric sampling. In *IEEE ICCV*, vol. 2, 1033–1038.

IJIRI, T., MĚCH, R., IGARASHI, T., AND MILLER, G. 2008. An example-based procedural system for element arrangement. *Comput. Graph. Forum 27*, 2, 429–436.

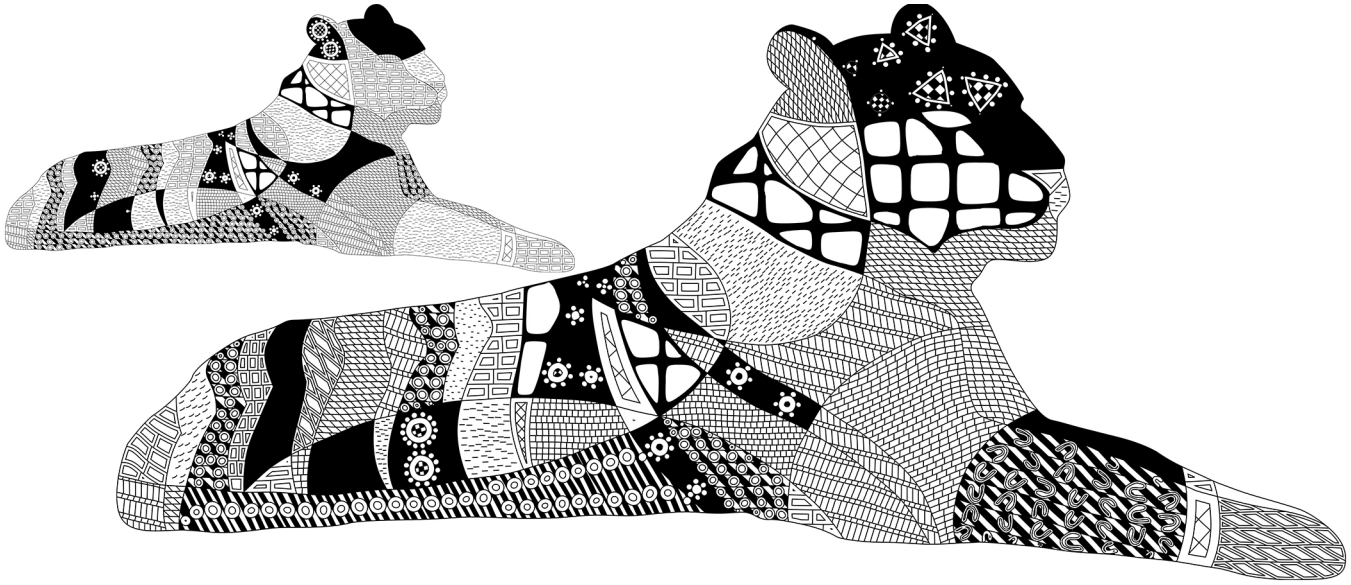


Figure 17: *Example tangle automatically generated by a group grammar (*top left*), that has been further edited by an artist (*bottom right*) using the features described in Section 6. Please refer to the supplemental video for the screen recorded editing session.*

KAZI, R. H., IGARASHI, T., ZHAO, S., AND DAVIS, R. 2012. Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *ACM SIGCHI '12*, 1727–1736.

KINDLMANN, G., AND WESTIN, C.-F. 2006. Diffusion tensor visualization with glyph packing. *IEEE T. Vis. Comput. Gr. 12*, 5 (Sept.), 1329–1336.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2d exemplars. *ACM Trans. Graph. 26*, 3.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (July), 777–786.

LI, Y., BAO, F., ZHANG, E., KOBAYASHI, Y., AND WONKA, P. 2011. Geometry synthesis on surfaces using field-guided shape grammars. *IEEE T. Vis. Comput. Gr. 17*, 2 (Feb.), 231–243.

LOI, H., HURTUT, T., VERGNE, R., AND THOLLOT, J. 2013. Discrete texture design using a programmable approach. In *ACM SIGGRAPH 2013 Talks*, SIGGRAPH '13, 43:1–43:1.

LU, J., BARNES, C., WAN, C., ASENTE, P., MECH, R., AND FINKELSTEIN, A. 2014. Decobrush: Drawing structured decorative patterns by example. *ACM Trans. Graph. 33*, 4, 90:1–90:9.

MA, C., WEI, L.-Y., AND TONG, X. 2011. Discrete element textures. *ACM Trans. Graph. 30*, 4, 62:1–62:10.

MEYER, M., BARR, A., LEE, H., AND DESBRUN, M. 2002. Generalized barycentric coordinates on irregular polygons. *J. Graph. Tools 7*, 1, 13–22.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *ACM SIGGRAPH '96*, 397–410.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *ACM SIGGRAPH '01*, 301–308.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 2012. *The algorithmic beauty of plants*. Springer Science.

PRUSINKIEWICZ, P., MÜNDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *ACM SIGGRAPH '01*, 289–300.

ROBERTS, R., AND THOMAS, M. 2012. *The book of Zentangles*. Zentangle Inc.

SCHWARZ, M., AND MÜLLER, P. 2015. Advanced procedural modeling of architecture. *ACM Trans. Graph. 34*, 4, 107:1–107:12.

SECORD, A. 2002. Weighted voronoi stippling. In *NPAR '02*, 37–43.

STINY, G. 1980. Introduction to shape and shape grammars. *Environment and planning B 7*, 3, 343–351.

STINY, G. 1982. Spatial relations and grammars. *Environ. Plan. B - Plan. Des. 9*, 1, 113–114.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH '01*, ACM, 347–354.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *ACM SIGGRAPH '00*, 479–488.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *Eurographics STAR*, 93–117.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. Graph. 22*, 3, 669–677.

XING, J., CHEN, H.-T., AND WEI, L.-Y. 2014. Autocomplete painting repetitions. *ACM Trans. Graph. 33*, 6, 172:1–172:11.

XING, J., WEI, L.-Y., SHIRATORI, T., AND YATANI, K. 2015. Autocomplete hand-drawn animations. *ACM Trans. Graph. 34*, 6, 169:1–169:11.

ZAINAB, A. 2009. Automatically mimicking unique hand drawn pencil lines. *Computers and Graphics 33*, 4, 496 – 508.

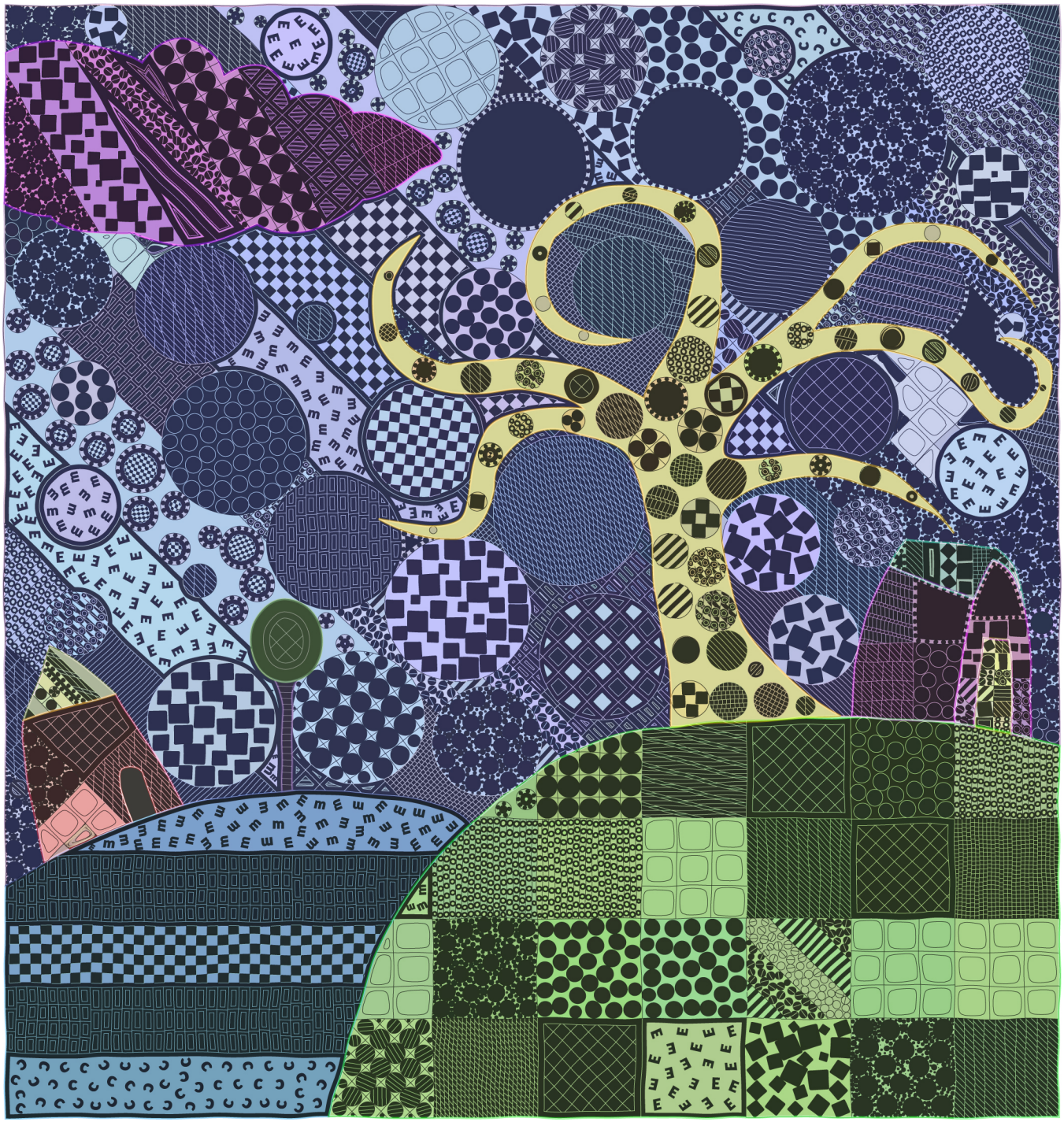


Figure 18: *Complex tangle, roughly* 17×17 *inches unscaled, generated by a group grammar that aggregates all the grammars used to produced the other figures in the paper. The shapes in the final tangle have been colored, in a fashion similar to naif art.*