Eurographics Symposium on Rendering (2023)
T. Ritschel and A. Weidlich (Editors)

FloralSurf: Space-Filling Geodesic Ornaments

V. Albano[1], F. A. Fanni[2], A. Giachetti[2] and F. Pellacini[3]

[1]Sapienza University of Rome
[2]University of Verona
[3]University of Modena and Reggio Emilia

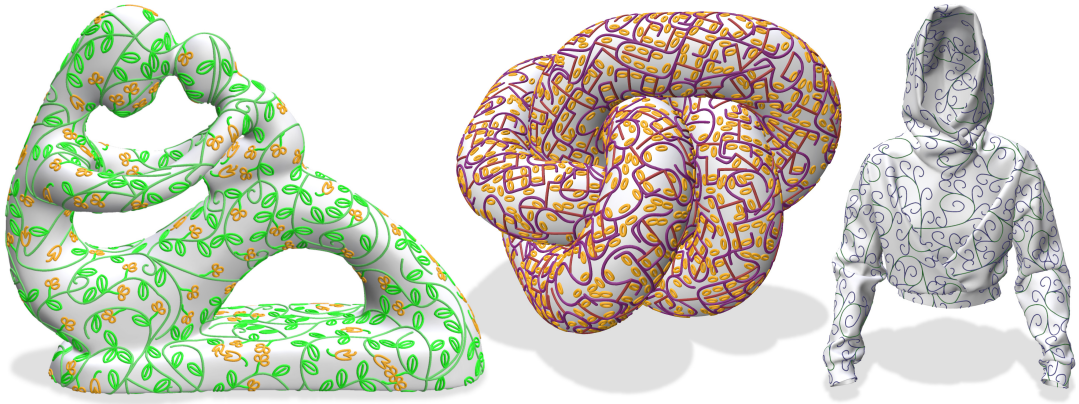


Figure 1: *Three examples of models decorated with our method. From left to right, a floral pattern is applied to the fertility model, an abstract pattern on a tight prime knot, and a spiral pattern on a hoodie. All these models pose significant challenges to traditional texturing techniques, either involving UV unwrapping or Euclidean space colonization with reprojection. Our method is instead able to handle them correctly, without introducing discontinuities or deformations, and following the topology of the surface even in the folds of the models.*

Abstract

We propose a method to generate floral patterns on manifolds without relying on parametrizations. Taking inspiration from the literature on procedural space-filling vegetation, these patterns are made of non-intersecting ornaments that are grown on the surface by repeatedly adding different types of decorative elements, until the whole surface is covered. Each decorative element is defined by a set of geodesic Bézier splines and a set of growth points from which to continue growing the ornaments. Ornaments are grown in a greedy fashion, one decorative element at a time. At each step, we analyze a set of candidates, and retain the one that maximizes surface coverage, while ensuring that it does not intersect other ornaments. All operations in our method are performed in the intrinsic metric of the surface, thus ensuring that the derived decorations have good coverage, with neither distortions nor discontinuities, and can be grown on complex surfaces.

In our method, users control the decorations by selecting the size and shape of the decorative elements and the position of the growth points. We demonstrate decorations that vary in the length of the ornaments' lines, and the number, scale and orientation of the placed decorations. We show that these patterns mimic closely the design of hand-drawn objects. Our algorithm supports any manifold surface represented as triangle meshes. In particular, we demonstrate patterns generated on surfaces with high genus, with and without borders and holes, and that can include a mixture of thin and large features.

CCS Concepts

• ***Computing methodologies*** → ***Graphics systems and interfaces; Shape modeling;***

1. Introduction

Procedural vector patterns are used to decorate vector drawings in applications such as Adobe Illustrator [Ado21], where they of-

DOI: 10.2312/sr.20231127

Figure 2: *A comparison of tracing ornaments on surfaces and applying them using parameterizations. We created a texture using our algorithm on a square and then convert it to a bitmap texture, which was then applied to the model, parameterized with [SC17] (center) and [Ble23] (right). We highlight in red some of the discontinuities and in purple the most evident distortions. We then ran our algorithm directly on the model with the same pattern. To keep the comparison fair and have a similar visualization, we render our results after rasterizing the curves into a texture (left).*

ten work as "vector textures" to fill large shapes in the drawings. Different algorithms are used to generate a variety of patterns: from simple tiling to stochastic patterns, from example-driven approaches [ŠBM*10, GJB*20, GBLM16, TWY*20] to space-filling decorations, such as tangles [SP16] and floral ornaments [WZS98, GALF17]. These vector patterns can be used to decorate 3D surfaces. The most common way to do so is to rasterize the pattern as a bitmap texture and then project it onto the surface via a parametrization. The distortions and seams that parametrizations necessarily exhibit on complex surfaces may be reasonable for some patterns, but become problematic for patterns that have a clear structure [NPP21]. Recent works have begun exploring the idea of generating structured procedural patterns directly on objects' surfaces.

In this work, we focus on floral ornaments, which are patterns composed of smooth curves that split like tree branches and fill the space without overlapping each other, accompanied by small decorations such as leaves and flowers. In the real world, floral ornaments are commonly used to decorate painted ceramics, etched metals and carved woods. When applying 2D floral ornaments to surfaces via parametrizations, the resulting patterns exhibit distortions and discontinuities of the ornaments lines and changes to the spatial distribution and orientation of the individual decorations, as shown in Figure 2. These artifacts are inherently present when using parametrization-based approaches and simple 2D Euclidean metrics in the texture space. For example, using overlapping charts with one local chart for each ornament would still introduce deformation on the larger charts, which could even amount to global parameterizations for large ornaments. Using more elaborate techniques than just Euclidean metrics, we could use a low-distortion parametrization presenting several cuts and employ jump maps to account for patch transitions. However, this would increase the computational complexity and still be an approximation of the geodesic metric. To precisely trace geodesics in the texture space we could employ similar techniques to [PKCH18] but it would be more complex than computing geodesic directly over the surface with no major benefits.

Our method synthesizes procedural floral ornaments directly on surfaces, without relying on a parametrization and is inspired by the space-colonization methods used in procedural vegetation [RLP07, RFL*05], that we port to the manifold setting. We generate geodesic floral ornaments by combining simple decorations, composed of geodesic Bézier curves, placed greedily on the surface until the surface is fully covered while ensuring that the added decorations never overlap the already-placed ornaments. Users control the final patterns by choosing the shape and size of the decorative elements and by defining on them a set of growth points from which the ornament can further split and grow. We guide the growth with a metric that attempts to maximize surface coverage while adopting a greedy algorithm to avoid the computational complexity of optimal packing problems, which are known to be NP-hard. In performing the growth, all operations are computed efficiently in the geodesic metric including tracing splines, computing distances, verifying intersections, etc.

We tested our method by generating detailed patterns on a variety of surfaces represented as triangle meshes. The resulting patterns vary in the length of the ornaments' lines, and the number, scale, and orientation of the decorations, and can closely match the design of hand-drawn patterns (see Figure 3). Our algorithm can grow patterns on surfaces with high genus, with and without borders and holes, and that can include a mixture of thin and large features.

Figure 3: *The hand-drawn vase on the left presents several continuous branches with leaves. We recreated a visually similar pattern with our method.*

2. Related work

Our method takes inspiration from ideas from different fields like 2D vector graphics synthesis, curve drawing on manifolds, space-filling pattern generation, and decoration packing.

Many methods have been proposed to generate vector graphics on the 2D Euclidean plane that target different pattern structures

and with different user controls. The interested reader should consult the survey [GAM*21].

A first class of algorithms generates patterns by examples. [MM10] presents a method for taking curves sketched by a user and automatically generating new curves that resemble the input shapes. [TWY*20] propose an example-based method to synthesize continuous curve patterns from exemplars. [LBW*14] propose a data-driven drawing system that allows designers to create highly structured patterns simply by choosing basic element styles and specifying an approximate overall path. Differently from these methods, we provide direct control to the user by specifying the decorative elements. [TWZ22] generate element-based patterns by explicit clustering to model larger element collections. These example-based methods cannot be directly extended to the manifold setting since they fundamentally assume that elements neighborhoods are stationary, which is not the case on manifolds.

Pattern generators differ in the manner in which patterns are controlled. [LHVT17] support full user control by defining patterns with a domain-specific language. [RSP22] estimates the parameters of procedural programs for vector patterns by optimization. [SKAM17] use vector fields to guide the placement of element-based decorations, creating streamlines, and deforming elements according to them. [GBLM16] propose a method for encoding distinct and valid pattern variations, allowing users to directly navigate the variation space.

Growing structures like vegetation patterns have often been generated using grammars and L-systems [PL12, STN*16], with optimization procedures used to estimate the program parameters [ŠBM*10, GJB*20]. The algorithmic generation of floral patterns has been introduced by [WZS98]. The authors observe that the generation of ornament differs from the growth of real plants in several significant ways and that the applicability of open L-systems in this context is limited. [GALF17] extend the previous approach by supporting global design constraints such as symmetry explicitly under artist control. Both these methods are limited to 2D ornaments since they assume that the local metric does not change and that the domain topology is an infinite plane when defining pattern repetitions, but neither of these assumptions is true in the manifold setting.

For this reason, we designed a space-filling algorithm inspired by the space colonization methods used for vegetation in [RLP07, RFL*05]. In these works, the iterative addition, in the Euclidean setting, of non-overlapping elements forms the final structure. In our approach, we add decorative elements directly on the manifold, while avoiding the overlap with other ornaments on the surface.

In our approach, decorations are specified as geodesic Bézier curves defined and traced directly on the surface. These curves have only recently become practical, thanks to the robust and efficient methods proposed in [SC20, MNPP22]. We also exploit the possibility of combining geodesic curves with the surface boolean operations presented in [RNP*22].

Extending the problem of creating a pattern to texture generation, many works have proposed methods to synthesize textures by example, even directly on surfaces. As noted in [WLKT09, AYD*18], most of these methods characterize the textures as local, i.e., each texture pixel depends only on the neighboring pixels, and as stationary, i.e., different regions of the texture (of some appropriate size) remain similar to each other. Early approaches, such as [EL99, WL00, Ash01, TZL*02], synthesize textures pixel-by-pixel by assigning to each output pixel the values of the input one with the most similar neighborhood. The quality of these methods is improved by synthesizing textures patch-by-patch [PFH00, LLX*01, EF01, KSE*03]. Many recent texture synthesis methods rely on Convolutional Neural Networks (CNNs). [GEB15] use the features of a pre-trained CNN to synthesize textures, while [Sne17] improves on texture resolution using a multi-scale approach. [TLH19] create irregular point sampling on a plane, and use it to create textures and terrains.

Texture synthesis methods may be also formulated on manifolds by using vector fields to specify texture orientation [PFH00, KCPS13]. The texture can then be generated by using scattered surface points to implement neighbor-based searches [WL01, Tur01], or possibly more sophisticated approaches [YHBZ01, ZG03, LH06]. [SGW06] use exponential maps to place decals; while this avoids global parameterizations, the exponential maps present discontinuities around small handles and holes. [PS06] describe a method for maze generation that can be applied on 3D surfaces using Euclidean distances to approximate the geodesic metric. In [DLL*15], an additional optimization step is employed to make the pattern structurally sound for fabrication. Similar structural goals are sought in [ZCT16, CXX*16], in which patterns are formed by packing small decorative elements and curves. [ROM*15, HWYZ20] are also element-based approaches to synthesize structures or autocomplete a partial texture. In [CXX*16], a local parameterization is used to perform the computations, and overlaps are also allowed in the final results. On the other hand, [ZCT16] avoid overlaps by placing scaled-down versions of the decorations and then iteratively scaling them up again, but approximates the geodesic metric with the tangent plane 2D Euclidean metric.

Compared to our work, most methods create textures with locality and stationary characteristics, meant to be tiled over the surface, with only some methods able to encode non-local features. The results suffer from distortions and discontinuities introduced by parametrizations. Lastly, the methods that rely on Euclidean distances can be unreliable on surfaces with high-frequency details. These limitations are the reason why we instead exploit geodesic operators to create decorative patterns directly on the surface instead of generating a texture and mapping it to the surface.

3. Space-Filling Geodesic Ornaments

In this section, we first give a formal definition of the terminology used in this paper. We then present the details of the algorithm and discuss the choices behind the design decisions.

3.1. Overview and definitions

The proposed algorithm combines ideas from space-filling techniques and element-based texturing while being formulated directly in the geodesic metric. Just like in space-filling methods, and inspired by procedural 2D ornaments and works on procedural vegetation, we greedily cover the whole domain, in our case a manifold

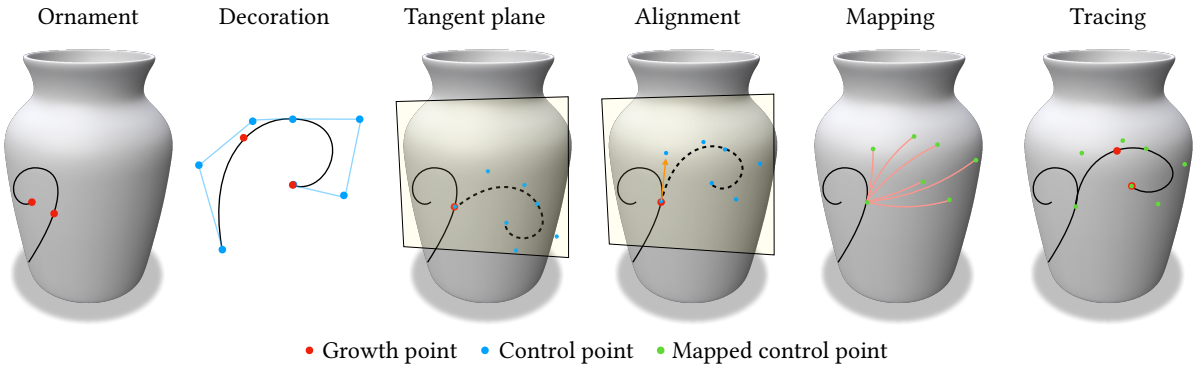


Figure 4: *A single step in the growth of an ornament. On the left the current state of the algorithm, with an already-placed ornament and two growth points inserted in the queue. One of them is selected and a decoration with a compatible tag is randomly chosen. The decoration is then aligned using the tangent plane of the current growth point and its associated growth direction. The control points of the Bézier comprising the decorations are mapped on the surface using straightest geodesics. Lastly, geodesic Bézier curves are traced on the surface.*

surface, by expanding a collection of *ornaments*, each of which defined by *decorative elements* composed of sets of *geodesic Bézier splines* (see Fig. 4). In the following, we give a formal definition of these concepts and introduce the pattern generation framework.

A *decorative element* is a simple drawing composed of a set of geodesic Bézier splines defined on a 2D reference frame. These decorations can be drawn on arbitrary manifolds by mapping the 2D control points to points on the surface, and tracing geodesic splines [MNPP22]. Decorative elements are associated with parameters determining material properties, size, and orientation.

On each decoration, we specify one or more *growth points* that indicate locations from which the ornament can further grow or split. Each growth point is associated with a tag, that selects the class of decorative elements that can grow from there, e.g. leaves or branches. At each growth point, we also associate a local frame that specifies the desired growth direction, for example aligned with the local spline tangent for continuous ornaments or angled on the side for branching structures.

An *ornament* on a 2D manifold is a continuous pattern generated by the iterative addition of decorative elements. It is logically a tree structure rooted by a starting decoration, with bifurcations corresponding to decorative elements with multiple growth points and leaves corresponding to elements without growth points or where the growth is stopped to avoid overlap.

3.2. Mapping decorative elements on surface meshes

To trace a decorative element on the surface, we map its 2D control points to points on the surface and trace geodesic splines on the manifold (see Fig. 4). We map the control points by considering a tangent frame whose origin coincides with the current growth point. We align such frame with respect to either the frame specified by the growth point or a global direction field. The former alignment replicated the continuous curves or branching structures defined in the decorative elements, while the latter gives global control over the direction of the ornaments. We map points between the 2D reference frame of the decorative element to the rotated tangent frame, and then trace straightest geodesics in the direction and length specified by the point location in the tangent frame origin. We considered also using a local parametrization, such as an exponential map, but found hard it to integrate it into our method since it is discontinuous around small handles in the mesh and significantly slower than tracing straightest geodesics.

3.3. Space filling procedure

Algorithm 1 contains the pseudo-code for the growth of the ornaments. For each seed point, we define a local frame on the mesh and place a decorative element at the corresponding location to start the ornament. We add then its growth points to a priority queue.

The priority can be used to make specific classes of decorative elements start to fill spaces after all those belonging to other classes cannot grow further, as shown in Figure 5. At each following step, the queue is processed extracting the next growth point and trying to add a decorative element compatible with it, according to the related tag. If a novel element can be successfully added, its growth

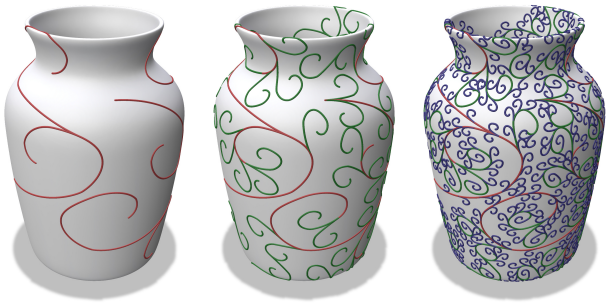


Figure 5: *Floral pattern grown at once from multiple seed points. From left to right: decorations of different scales are placed at different times in the pattern growth using a priority queue.*

Algorithm 1: Pseudo code of our proposed algorithm.

```
def trace_ornaments(params):
  # draws one or more ornaments
  # depending on the number of seeds
  ornaments = init_ornaments()
  queue = init_queue(params.num_seeds,
    params.align_to_field)
  while queue is not Empty:
    decoration, splines =
      choose_decoration(queue.pop())
    if deco is None: continue
    append_splines(ornaments, splines)
    update_geodesic_distances(splines)
    append_growthpoints(queue, decoration)
    update_intersections(spline)
  return ornaments

def choose_decoration(growth_point):
  candidates = []
  for _ in range(trials):
    decoration = random_decoration(growth_point)
    control_pts = map_decoration(decoration)
    splines = trace_geodesic_splines(control_pts)
    if intersect_ornaments(splines): continue
    candidates.append((decoration, splines))
  if candidates == []: return None, None
  scores = score_splines(candidates)
  return candidates[argmax(scores)]

def map_decoration(...):
  # compute angle according to user options,
  # direction field and parent direction
  if params.align_to_field:
    rot_angle = field_direction
  else:
    rot_angle = parent_tangent
  for control_pt in decoration.control_pts:
    rotated_point = rotate(control_pt, rot_angle)
    trace_straightest_geodesic(start_point,
        rotated_point)
  return spline
```

points are added to the queue and the procedure is iterated until the queue is empty.

At each step, we consider a list of candidates with tags compatible with the growth point tag. For each candidate, we trace the corresponding splines on the surface and test for intersection with the already-placed ornaments. Of all non-intersecting candidates, we pick the one that is, on average, farthest away from the already placed decorations, as is done in best candidate sampling [Mit91], to improve surface coverage. More specifically, we pick the non-intersecting decoration that maximizes the following metric

$$S = \frac{\sum_{i=1}^{n} D(P_n)}{n}$$

where n is the number of points P in the decoration, and the function D is the geodesic distance from the points of already placed decorations (which we keep track of throughout the whole procedure). We chose this metric since it guides the algorithm to select

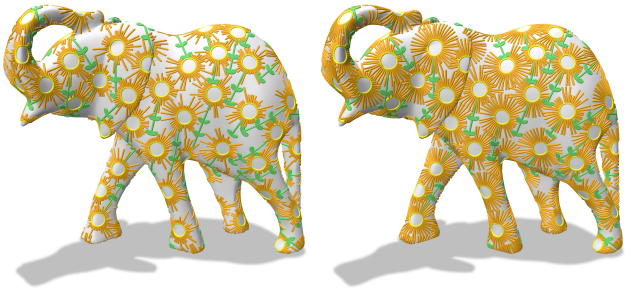


Figure 6: *A dandelion-like pattern. The circles are placed first and from there the stem and petals grow to fill the remaining space. There are three possible sizes for the petals, and selecting the first non-intersecting candidate decoration (on the left) yields poor coverage of the mesh. Instead (on the right), the best candidate sampling and our space-filling metric guide the algorithm to almost always select the longest petal possible.*

decorations that split the empty space well. While checking for candidates, we could consider all possible ones, but that would lead to high computation times. Instead, we pick only ten candidates at random to keep execution time constant. Another possibility would be to perform the assignment in a purely greedy fashion, i.e. considering only one candidate. In our experiments, we found that approach to produce low-quality patterns, as shown in Figure 6.

Floral patterns are generally comprised of multiple ornaments, since a single one may not cover the whole surface well, depending on the choice of input decorations and the surface shape. To cover the whole surface, we grow multiple patterns, either sequentially or concurrently, as shown in Figure 7. We grow patterns sequentially, i.e. one after another, by running the ornament growth algorithm (Algorithm 1) multiple times with a single seed. For better surface coverage, we select as the starting point of each ornament the point with the largest geodesic distance to the ornaments already placed. We grow patterns concurrently, by initializing the growth procedure (Algorithm 1) with multiple seeds, again using a best-candidate approach in the geodesic metric. Internally, these seed points are effectively growth points, not different from the ones inserted by the growing decorations. In general, growing patterns sequentially favors better surface coverage, while growing patterns concurrently generates ornaments of similar lengths. In both methods, the orientation of the patterns at their starting locations are either random or aligned with a global direction field (see Figure 8).

By setting the growth procedures with the related options, and selecting for them different sets of ornaments with optional placement priority, it is possible to draw on complex shapes a broad variety of patterns, as shown in Section 4. These controls are similar to current practice for procedural methods such as grammar-based approaches [SW15] and industry-standard workflows in Blender and Houdini.

3.4. Discussion

In the previous subsection, we discussed most of the design decisions that motivate our algorithm. But we did so under the as-

Figure 7: *An ornament, whose decorations are not designed to branch, will not fill the surface (left). To cover the whole surface, we can either grow multiple ornaments concurrently by starting from multiple locations at once (center), or grow multiple ornaments sequentially one after the other (right). We note that concurrent growth keeps the ornaments more similar in length, while sequential growth gives better surface coverage.*

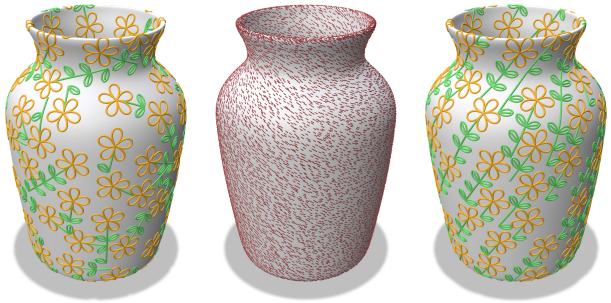


Figure 8: *Ornaments may start with random alignments (left) or have a coherent orientation (right) that is aligned with a global direction field (middle).*

sumption of a formulation based on space-filling with a fixed set of decorations. In this subsection, we revisit that assumption and compare it to possible alternatives.

The main motivation for using a fixed set of decorations is to provide direct user control during synthesis. The shape of the decorations defines the overall look of the final pattern. The placement of growth points and the use of tags directly control the growth on the mesh. To the best of our knowledge, no other work gives the same level of direct control as using predefined decorations.

The use of a predefined set of decorations and their guidance by defining growth points and tags may sound similar to shape grammars that guide shape synthesis with tagged rules. The main difference of this work is that grammars cannot express space-filling constraints since they define substitution rules that are applied locally in time, rather than globally as constraints on the synthesis. In fact, our algorithm does not use user-specified rules, but only branching positions and labels as in procedural vegetation.

Our method can be seen as a variant of packing a specified domain with as many shapes as possible. This class of problem is known to be NP-hard, so most algorithms presented are approximations that differ in the constraints they pose on the final solution. In the graphics literature, two main classes of algorithms have been explored. Greedy packing, possibly with backtracking, attempts to fill the space one decoration at a time. Optimization-based methods attempt to place all decorations at once by using proxy shapes, such as circles or rectangles, to perform the initial placement and then deform the decorations to minimize overlap. In our work, we favor a greedy placement approach since it supports all decoration shapes, regardless of whether they can be approximated with simple proxies, and since it does not deform the original decorations, thus maintaining direct user control. An optimization-based method might have also worked, but no current methods we know support arbitrary and undeformed decoration shapes.

3.5. Implementation

We implemented our prototype in Python using wrapped C++ libraries for the geodesic computation kernels. Our algorithm is independent of the choice of implementation for most computation kernels. Our choice was guided by the availability of implementations, their robustness, and their interoperability.

Our method depends entirely on a fast and robust implementation of geodesic Bézier splines. We use cubic Bézier curves since they are the foundation of 2D vector graphics, and since we can easily use 2D decorations from available libraries. We represent Bézier curves with control points specified as pairs of triangle identifier and barycentric coordinates. To perform further computation, we approximate Bézier curves as lists of segments with vertices represented intrinsically as triangle identifiers and barycentric coordinates pairs. In our prototype, we use the geodesic Bézier implementation from [MNPP22] that we found to be reliable for our needs. Since Bézier curves are tessellated into segments whose coordinates are referred intrinsically to each triangle, we perform intersection tests one triangle at a time. In each triangle, the intersections are expressed in barycentric coordinates, so they can be resolved as 2D segment-to-segment intersections. We speed up computation by first determining the list of segments that cross each triangle, which can be done trivially using triangle ids, and then processing the intersections in each triangle independently. For highly tessellated surfaces, this method is linear in the number of segments.

We map the 2D control points of each decoration to intrinsic points on the surface by using straightest geodesics traced from the tangent space of the current growth point. We do not depend on a specific implementation, provided that it supports expressing all computation in intrinsic coordinates. In our prototype, we use the implementation from [SC*19].

We compute geodesic distances using the heat method [CWW13] using the implementation provided in [SC*19]. We chose the heat method since it is efficient for non-changing meshes and accurate enough for our application. Modern geodesic solvers might provide more accurate and efficient alternatives [NPP21, TBK21]. In our experiments, however, we found the time used to compute geodesic distances negligible with respect to the computation of the geodesic Béziers.

We compute the smoothest direction field using the method pre-

sented in [KCPS13] and implemented in [SC*19], as it provides a simple initialization for the subsequent operations. More direct control may be achieved with different methods, although this choice is mostly orthogonal to our work.

3.6. Limitations

Our main limitation stems from the greedy assumption of our algorithm. Since we are solving a packing problem, that is NP-hard, we cannot guarantee good coverage of the surface, nor that we can grow ornaments that are long with respect to the surface size. This limitation is shared with all other packing methods. Optimization methods do not have this limitation, but in turn may arbitrarily deform the decorations to achieve good coverage. We mitigate this problem by growing multiple ornaments. In our tests, we found this problem to be only relevant when the decorations' sizes are similar to the size of the whole surface, which is generally not desirable since decorations are meant to be small.

A second limitation is that our execution speeds are not yet interactive, as we will show in the next section. This is partly due to our unoptimized Python implementation that is good enough for exploring the problem, but not sufficiently fast to be executed in real-time. More significantly though, a very large number of geodesic queries are executed to generate a full pattern. Packing problems are notoriously slow in the Euclidean setting, and performing all computations in the geodesic metric, including spline tessellation, curve intersection, and distance computation, makes them even slower. We expect that the speed of our algorithm will improve significantly as better geodesic computation kernels are developed and maybe ported to massively parallel architectures. Furthermore, performance is limited by the high number of triangles, and by the fine tessellation of splines, which were used to ensure a good approximation of all the geodesic functions. To improve the user experience, we could use coarser tessellations while editing, and use finer ones for final rendering, as typical in subdivision methods.

4. Results

Throughout this paper, we have shown several results created with our method. We include further results in Figure 9. The variety of these results shows that our algorithm is general and can be easily controlled by changing input decorations. Indeed, we demonstrate long continuous ornaments, branching floral and abstract patterns, and synthesis at multiple scales. In particular, in Figure 9 we show a floral pattern generated from 4 seeds in parallel, with 50 more flowers added in a second step (9a), a minimal and an abstract pattern grown by a single seed (9b and 9c), a pattern made with a parallel growth from a pair of seeds 20 times (9d), and finally, one pattern created by growing thousands of seeds in parallel (9e).

In terms of input surfaces, we support smooth and corrugated meshes and surfaces with low and high genus, with or without boundaries. Our only assumption is that the underlying surface is manifold so that decoration placement is well-defined. This generality shows that our method is useful in practice to decorate most surfaces.

Table 1 reports summary statistics for the examples in this paper. In terms of the generated patterns, our results have on average two thousand decorations. Our most complex result in terms of the number of decorations is the dress image in 9e, where we place more than 7 thousand decorations, requiring almost 3 million geodesic segments to represent the 22 thousand geodesic Bézier splines applied.

The reported times are measured on a single-thread implementation running on a 3.8GHz CPU. Most of the execution times are in the 100 to 500 seconds range, with only a few outliers, which makes our prototype useful for offline texturing and modeling applications. On the other hand, initialization times required for the creation of the geodesic solvers are on average less than five seconds. Overall, we believe that by lowering the implementation to C++ and using multiple threads, we can easily cut computation times by more than a factor of 10, since the algorithm is trivially parallelizable. At the same time, we should note that our algorithm performs a remarkably large number of geodesic operations since we focus on modeling highly detailed patterns. For example, our most complex result in terms of the number of operations is the hoodie model in Figure 1, where we trace one million straightest geodesics and almost 300 thousand geodesic Bézier splines.

Furthermore, the ornaments always achieve good coverage of the mesh, except in the vase of 9b, where the decorations were specifically designed not to cover the whole surface. Considering that we are packing one-dimension lines on a surface, we consider only an approximate metric of surface coverage by reporting the average distance from each mesh vertex to the closest point of the pattern. We report also the area of the base mesh since the chosen metric depends on it. If we consider the square root of the surface area as a characteristic size of the mesh, we can easily compare it with the average vertex to decoration distance. The median value we find in doing so is 0.007, where the worst possible value one could achieve on a unit sphere by placing a single point is approximately 0.44, supporting our claim of good coverage.

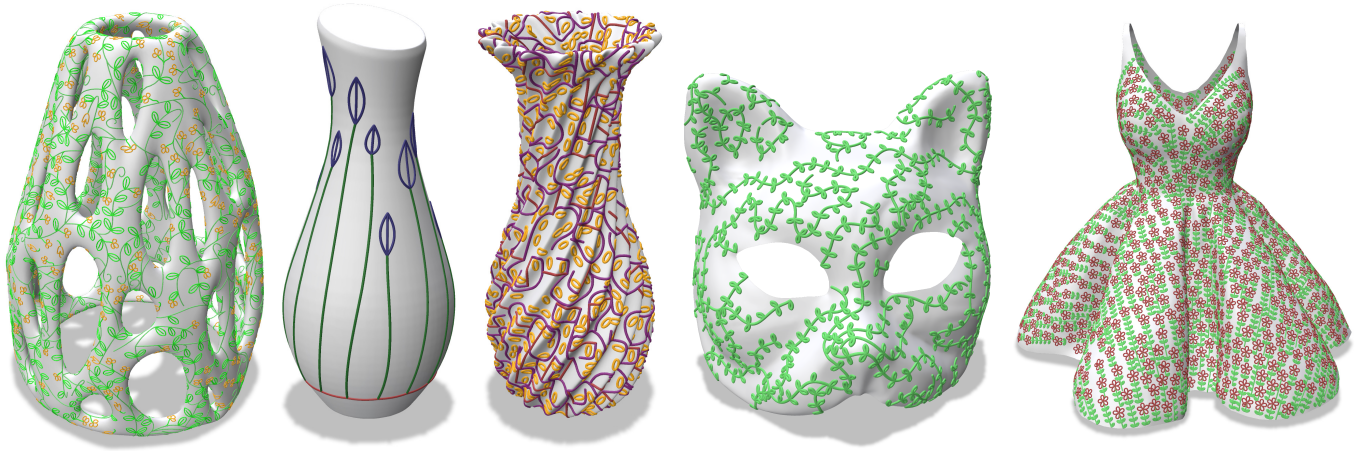
5. Conclusions

In summary, this paper presented *FloralSurf*, an algorithm for generating geodesic space-filling ornaments directly on the surface. *FloralSurf* takes as input a set of deformed, defined by cubic Bézier splines with specified endpoints, and generates ornaments by placing decorations greedily one after the other. In *FloralSurf*, artists control the final pattern by choosing a decoration set, and can apply the pattern on any manifold surfaces, including surfaces with high genus and open boundaries, without requiring a parametrization.

In the future, we plan to further speed up our algorithm by investigating geodesic computational kernels that run efficiently on massively parallel hardware, and provide more direct control by integrating interactive sketching to guide the growth.

Acknowledgments

This study was partially partially carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza



(a) (b) (c) (d) (e)

Figure 9: *More patterns generated by* FloralSurf*, including results on high genus and open boundary meshes.*

Model	Init time	Exec time	Mesh triangles	Decos	Splines	Growth points	Iterations	Straightest geodesics	Splines traced	Geodesic segments	Average distance	Mesh surface
Fig. 1, left	4.0 s	251.3 s	80k	815	2633	1831	1885	198250	57636	367505	0.01155	3.02
Fig. 1, center	2.4 s	526.6 s	60k	4568	7287	9454	9455	574412	156608	964953	0.00887	6.10
Fig. 1, right	13.0 s	2884.8 s	200k	1271	6355	5882	5936	1127840	296800	857887	0.00911	3.35
Fig. 3	2.6 s	162.3 s	60k	1937	3513	2347	2407	129128	43330	465139	0.01428	2.93
Fig. 5	5.5 s	1458.4 s	100k	1458	7290	6733	6745	1281550	337250	956718	0.01153	4.17
Fig. 6, right	4.8 s	460.5 s	100k	4340	5210	5133	5237	208070	67970	720049	0.00593	2.35
Fig. 7, left	5.7 s	35.9 s	100k	253	464	293	294	16212	5440	65725	0.10045	4.17
Fig. 7, center	5.5 s	157.4 s	100k	877	1545	1353	1403	74384	24970	218799	0.02004	4.17
Fig. 7, right	5.5 s	145.1 s	100k	889	1586	1216	1266	66678	22400	223602	0.01821	4.17
Fig. 8, left	5.5 s	182.9 s	100k	652	2163	852	1012	80440	29480	314214	0.01575	4.17
Fig. 8, right	5.5 s	215.5 s	100k	843	2527	1083	1243	96610	34870	362598	0.01457	4.17
Fig. 9a	3.0 s	496.7 s	60k	2331	7125	5440	5494	562586	163826	949045	0.00768	2.90
Fig. 9b,	2.9 s	4.2 s	60k	44	72	45	46	2115	705	13587	0.04659	1.05
Fig. 9c	6.2 s	493.3 s	120k	1995	3206	4159	4160	254972	69323	454419	0.00760	2.59
Fig. 9d,	1.3 s	45.5 s	40k	804	1450	1016	1056	52797	17116	194098	0.01588	1.31
Fig. 9e,	2.3 s	964.8 s	60k	7592	22663	9432	10947	838589	300176	2936602	0.00649	2.39

Table 1: *Summary statistics for the results in the paper. For each model, we list the initialization time in seconds to generate all the geodesic solvers, the execution time in seconds to create the ornaments, the number of triangles of the base model, the number of decorations comprising the ornaments, the number of splines comprising the decorations, the number of growth points, the number of iterations performed by the algorithm, the number of straight geodesics performed, the number of Bézier splines constructed (including the discarded ones), the number of geodesic segments necessary to represent the ornaments, the average distance from the vertices to the closest ornament, and the base mesh surface. Note that we are testing 10 decorations at each growth point.*

(PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS00000043). For the models, we thank: ManuelDerErste for the corrugated vase, lahbibforce for the simple, tall vase, fablabuv for the vase with holes, designbynumbers for the prime knot, Angelina Scher for the dress, Marashe for the hoodie, Greg Zaal for the elephant, James Ray Cock for the wide vase, and terpsichore for the cat mask.

References

[Ado21] Adobe illustrator, 2021. URL: https://adobe.com/products/illustrator. 1

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *Proc. of Interactive 3D Graphics* (2001), p. 217–226. 3

[AYD*18] AKL A., YAACOUB C., DONIAS M., DA COSTA J.-P., GERMAIN C.: A survey of exemplar-based texture synthesis methods. *Computer Vision and Image Understanding 172* (2018), 12–24. 3

[Ble23] BLENDER F.: Blender. https://www.blender.org, 2023. 2

[CWW13] CRANE K., WEISCHEDEL C., WARDETZKY M.: Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph. 32*, 5 (2013). 6

[CXX*16] CHEN W., XIA X., XIN S., XIA Y., LEFEBVRE S., WANG W.: Synthesis of Filigrees for Digital Fabrication. *ACM Trans. Graph. 35*, 4 (2016). 3

[DLL*15] DUMAS J., LU A., LEFEBVRE S., WU J., DICK C.: By-example synthesis of structurally sound patterns. *ACM Trans. Graph. 34*, 4 (jul 2015). 3

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *Proc. of ACM SIGGRAPH* (2001), p. 341–346. 3

[EL99] EFROS A., LEUNG T.: Texture synthesis by non-parametric sampling. In *Proc. of ICCV* (1999), vol. 2. 3

[GALF17] GIESEKE L., ASENTE P., LU J., FUCHS M.: Organized order in ornamentation. In *Proc. of Symp. on Comp. Aesthetics* (2017), CAE '17. 2, 3

[GAM*21] GIESEKE L., ASENTE P., MĚCH R., BENES B., FUCHS M.: A survey of control mechanisms for creative pattern generation. *Computer Graphics Forum 40*, 2 (2021), 585–609. 3

[GBLM16] GUERRERO P., BERNSTEIN G., LI W., MITRA N. J.: Patex: Exploring pattern variations. *ACM Trans. Graph. 35*, 4 (jul 2016). 2, 3

[GEB15] GATYS L. A., ECKER A. S., BETHGE M.: Texture synthesis using convolutional neural networks. In *Proc. of NIPS* (2015), p. 262–270. 3

[GJB*20] GUO J., JIANG H., BENES B., DEUSSEN O., ZHANG X., LISCHINSKI D., HUANG H.: Inverse procedural modeling of branching structures by inferring l-systems. *ACM Trans. Graph. 39*, 5 (2020). 2, 3

[HWYZ20] HSU C.-Y., WEI L.-Y., YOU L., ZHANG J. J.: Autocomplete element fields. In *Proc. of SIGCHI* (2020), p. 1–13. 3

[KCPS13] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Globally optimal direction fields. *ACM Trans. Graph. 32*, 4 (2013), 1–10. 3, 7

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3 (jul 2003), 277–286. 3

[LBW*14] LU J., BARNES C., WAN C., ASENTE P., MECH R., FINKELSTEIN A.: Decobrush: Drawing structured decorative patterns by example. *ACM Trans. Graph. 33*, 4 (2014), 1–9. 3

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Trans. Graph. 25*, 3 (2006), 541–548. 3

[LHVT17] LOI H., HURTUT T., VERGNE R., THOLLOT J.: Programmable 2d arrangements for element texture design. *ACM Trans. Graph. 36*, 4 (2017), 1. 3

[LLX*01] LIANG L., LIU C., XU Y.-Q., GUO B., SHUM H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph. 20*, 3 (2001), 127–150. 3

[Mit91] MITCHELL D. P.: Spectrally optimal sampling for distribution ray tracing. In *Proc. of ACM SIGGRAPH* (1991), p. 157–164. 5

[MM10] MERRELL P., MANOCHA D.: Example-based curve synthesis. *Computers & Graphics 34*, 4 (2010), 304–311. 3

[MNPP22] MANCINELLI C., NAZZARO G., PELLACINI F., PUPPO E.: B/surf: Interactive bézier splines on surfaces. *IEEE Trans. Vis. Comp. Graph* (2022). 3, 4, 6

[NPP21] NAZZARO G., PUPPO E., PELLACINI F.: geoTangle: Interactive design of geodesic tangle patterns on surfaces. *ACM Trans. Graph. 41*, 2 (2021), 12:1–12:17. 2, 6

[PFH00] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. In *Proc. of ACM SIGGRAPH* (2000), p. 465–470. 3

[PKCH18] PRADA F., KAZHDAN M., CHUANG M., HOPPE H.: Gradient-domain processing within a texture atlas. *ACM Trans. Graph. 37*, 4 (2018). 2

[PL12] PRUSINKIEWICZ P., LINDENMAYER A.: *The algorithmic beauty of plants*. Springer Science & Business Media, 2012. 3

[PS06] PEDERSEN H., SINGH K.: Organic labyrinths and mazes. In *Proc. of NPAR* (2006), p. 79–86. 3

[RFL*05] RUNIONS A., FUHRER M., LANE B., FEDERL P., ROLLAND-LAGAN A.-G., PRUSINKIEWICZ P.: Modeling and visualization of leaf venation patterns. In *ACM SIGGRAPH 2005 Papers*. 2005, p. 702–711. 2, 3

[RLP07] RUNIONS A., LANE B., PRUSINKIEWICZ P.: Modeling trees with a space colonization algorithm. *NPH 7*, 63-70 (2007), 6. 2, 3

[RNP*22] RISO M., NAZZARO G., PUPPO E., JACOBSON A., ZHOU Q., PELLACINI F.: Boolsurf: Boolean operations on surfaces. *ACM Trans. Graph. 41*, 6 (2022). 3

[ROM*15] ROVERI R., ÖZTIRELI A. C., MARTIN S., SOLENTHALER B., GROSS M.: Example based repetitive structure synthesis. In *Proc. of SGP* (2015), p. 39–52. 3

[RSP22] RISO M., SFORZA D., PELLACINI F.: pop: Parameter optimization of differentiable vector patterns. *Computer Graphics Forum 41*, 4 (2022), 161–168. 3

[ŠBM*10] ŠT'AVA O., BENEŠ B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum 29*, 2 (2010), 665–674. 2, 3

[SC17] SAWHNEY R., CRANE K.: Boundary first flattening. *ACM Trans. Graph. 37*, 1 (2017). 2

[SC*19] SHARP N., CRANE K., ET AL.: geometry-central, 2019. www.geometry-central.net. 6, 7

[SC20] SHARP N., CRANE K.: You can find geodesic paths in triangle meshes by just flipping edges. *ACM Trans. Graph. 39*, 6 (2020), 249:1–15. 3

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph. 25*, 3 (2006), 605–613. 3

[SKAM17] SAPUTRA R. A., KAPLAN C. S., ASENTE P., MECH R.: Flowpak: Flow-based ornamental element packing. In *Graphics Interface* (2017), pp. 8–15. 3

[Sne17] SNELGROVE X.: High-resolution multi-scale neural texture synthesis. In *SIGGRAPH Asia 2017 Technical Briefs* (2017). 3

[SP16] SANTONI C., PELLACINI F.: Gtangle: A grammar for the procedural generation of tangle patterns. *ACM Trans. Graph. 35*, 6 (dec 2016). 2

[STN*16] SHAKER N., TOGELIUS J., NELSON M. J., TOGELIUS J., SHAKER N., DORMANS J.: Grammars and l-systems with applications to vegetation and levels. *Procedural Content Generation in Games* (2016), 73–98. 3

[SW15] SCHWARZ M., WONKA P.: Practical grammar-based procedural modeling of architecture. In *SIGGRAPH Asia 2015 Courses* (2015). 5

[TBK21] TRETTNER P., BOMMES D., KOBBELT L.: Geodesic distance computation via virtual source propagation. *Computer Graphics Forum 40*, 5 (2021), 247–260. 6

[TLH19] TU P., LISCHINSKI D., HUANG H.: Point pattern synthesis via irregular convolution. *Computer Graphics Forum 38*, 5 (2019), 109–122. 3

[Tur01] TURK G.: Texture synthesis on surfaces. In *Proc. of ACM SIGGRAPH* (2001), p. 347–354. 3

[TWY*20] TU P., WEI L.-Y., YATANI K., IGARASHI T., ZWICKER M.: Continuous curve textures. *ACM Trans. Graph. 39*, 6 (2020), 1–16. 2, 3

[TWZ22] TU P., WEI L.-Y., ZWICKER M.: Clustered vector textures. *ACM Trans. Graph. 41*, 4 (2022), 1–23. 3

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.-Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Graph. 21*, 3 (2002), 665–672. 3

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *Proc. of ACM SIGGRAPH* (2000), p. 479–488. 3

[WL01] WEI L.-Y., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. In *Proc. of ACM SIGGRAPH* (2001), p. 355–360. 3

[WLKT09] WIE L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the Art in Example-based Texture Synthesis. In *EG STAR* (2009), Pauly M., Greiner G., (Eds.). 3

[WZS98] WONG M. T., ZONGKER D. E., SALESIN D. H.: Computer-generated floral ornament. In *Proc. of ACM SIGGRAPH* (1998), pp. 423–434. 2, 3

[YHBZ01] YING L., HERTZMANN A., BIERMANN H., ZORIN D.: Texture and shape synthesis on surfaces. In *Rendering Techniques 2001* (2001), pp. 301–312. 3

[ZCT16] ZEHNDER J., COROS S., THOMASZEWSKI B.: Designing structurally-sound ornamental curve networks. *ACM Trans. Graph. 35*, 4 (2016). 3

[ZG03] ZELINKA S., GARLAND M.: Interactive Texture Synthesis on Surfaces Using Jump Maps. In *Eurographics Workshop on Rendering* (2003). 3