PAVEL: Decorative Patterns with Packed Volumetric Elements

FILIPPO ANDREA FANNI, University of Verona, Italy
FABIO PELLACINI, Sapienza University of Rome, Italy
RICCARDO SCATENI, University of Cagliari, Italy
ANDREA GIACHETTI, University of Verona, Italy

Fig. 1. Three examples of objects decorated with packed volumetric elements generated by *PAVEL*. On the left, the decoration of the Fertility digital model, in the center, a bracelet, and on the right, a plague doctor masque, inspired by the Venetian carnival tradition. In *PAVEL*, decorations are placed on a surface in an overlapped configuration. We then deform the decorative elements in a manner that mimics the plastic deformations of real-world artworks. For each model, we show the base shape, in blue, with the positions for the decorative elements. The placement of decorations is automatic in the first two examples and can also be done manually for better artistic control, as shown in the latter example.

Many real-world hand-crafted objects are decorated with elements that are packed onto the object's surface and deformed to cover it as much as possible. Examples are artisanal ceramics and metal jewelry. Inspired by these objects, we present a method to enrich surfaces with packed volumetric decorations. Our algorithm works by first determining the locations in which to add the decorative elements and then removing the non-physical overlap between them while preserving the decoration volume. For the placement, we support several strategies depending on the desired overall motif. To remove the overlap, we use an approach based on implicit deformable models creating the qualitative effect of plastic warping while avoiding expensive and hard-to-control physical simulations. Our decorative elements can be used to enhance virtual surfaces, as well as 3D-printed pieces, by assembling the decorations onto real surfaces to obtain tangible reproductions.

This work was partially supported by the MIUR Excellence Departments (Dipartimenti di Eccellenza) project 2018-2022.
Authors' addresses: F. A. Fanni and A. Giachetti, Dept. of Computer Science, University of Verona, Verona, Italy; emails: {filippoandrea.fanni, andrea.giachetti}@univr.it; F. Pellacini, Dept. of Computer Science, Sapienza University of Rome, Rome, Italy; email: pellacini@di.uniroma1.it; R. Scateni, Dept. of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy; email: riccardo@unica.it.

0730-0301/2022/01-ART19 $15.00
https://doi.org/10.1145/3502802

CCS Concepts: • **Computing methodologies** → **Volumetric models**; **Mesh models;**

Additional Key Words and Phrases: Shape deformation, artistic design

ACM Reference format:
Filippo Andrea Fanni, Fabio Pellacini, Riccardo Scateni, and Andrea Giachetti. 2022. PAVEL: Decorative Patterns with Packed Volumetric Elements. *ACM Trans. Graph.* 41, 2, Article 19 (January 2022), 15 pages.
https://doi.org/10.1145/3502802

1 INTRODUCTION

The quest for representing reality is one of the main goals of computer graphics since its origins. Due to the advances in modeling, rendering, and AI, it is often possible to virtually reproduce the look of real-world objects, characters, and scenes in a manner that is indistinguishable from reality. Recent advances in digital manufacturing allow us to physically reproduce realistic objects that were initially modeled as digital shapes. This is becoming an important field of application for 3D geometry processing algorithms and techniques.

The difference between a model that appears realistic and one that does not is often the complex surface details typical of real-world objects. In graphics, surfaces are often enriched by varying their material properties, texture maps, or generating relief patterns. As pointed out in a recent survey [Zhang et al. 2019], most of the current methods used to generate reliefs patterns work in image-space, are applied mainly on planar objects, and cannot

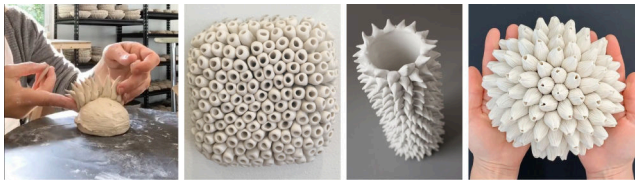


Fig. 2. *Left:* An artist creating volumetric decoration on a surface by adding plastic elements over a base surface. *Center and right:* Examples of hand-made pottery created with such a method (Courtesy of Heather Knight).

reproduce details coming from 3D decorative elements placed onto the surface.

In this work, we present *PAVEL*, a method for decorating surfaces with *pa*cked *v*olumetric *el*ements. Our method is inspired by hand-crafted artworks, shown in Figure 2, that are decorated by manually packing elements onto objects. Each decoration is deformed to better cover the surfaces. These decorations are often found in hand-made ceramic, metal jewelry, and Murano's glass. Figure 1 shows examples of surfaces decorated with *PAVEL* to mimic these styles.

Current methods cannot reproduce these decorations. Techniques based on surface displacement, such as procedural methods and digital sculpting, cannot do it, since they do not model local surface topology changes. Element-based textures cannot be used either, since they are generally defined in Euclidean spaces, while we model decorations on manifolds, and since they do not support deformation that maintains the elements volumes, which is required to achieve realism.

PAVEL works in two stages. We first place decorations onto the surfaces with considerable overlap between them. In our implementation, we support both the manual placement of the elements that mimics exactly the real-world process and the automatic generation of different kinds of patterns of overlapped elements. Decorations overlap to cover the surface as much as desired by the user. We then deform the decoration to remove the overlap between them. In doing so, we preserve the volume while deforming to simulate real-world behavior. These two stages produce results that are visually similar to real-world decorations, as shown throughout the article. In *PAVEL*, we make two critical choices to make computation feasible.

First, we use point sampling to place the decorations' centroids instead of using packing algorithms. In general, packing algorithms are just heuristics, since packing is NP-hard in the Euclidean domain and just as hard on the manifold. Point sampling is significantly more efficient, by a couple of orders of magnitudes, and just as reliable in our domain. This choice favors objects having a radial symmetry but does not entirely exclude other entities, as we show in the results.

Second, we simulate the plastic deformation of the elements pushed over the shape with an efficient implicit method inspired by Gascuel [1993]. We start by removing the overlapped parts of the elements and then make them grow in the free space around until the original volume is recovered. By using a specifically designed propagation field and a fast marching approach it is possible to obtain deformations resembling those appearing in real plastic elements. A similar effect cannot be obtained easily with a physics simulator, as it is necessary to avoid initial overlaps, for numerical robustness, apply precise forces, to control the deformation, and set non-trivial parameters and constraints, to customize the simulator to our needs (see Section 8).

The main contributions of our work are:

- an overall method for decorating surfaces with packed volumetric elements supporting automatic and manual element decorations, and automatic removal of decoration overlap; to the best of our knowledge, this is the first packing method that works on 2-manifold directly; to the best of our knowledge, the decorated surfaces could not be created with any other automatic method and would require hours of manual work if sculpted manually [Santoni et al. 2016];
- an efficient procedure to remove intersections and recover elements' volume that provides a visually effective, even if not physically exact, simulation of a plastic, volume-preserving deformation;
- a user evaluation showing that the designed shapes are really similar to the hand-crafted artworks and can be realized in an efficient manner;
- a practical demonstration of a decorative shell for an existing 3D object designed with *PAVEL*, decomposed into printable parts, manufactured and attached to the base shape.

2 RELATED WORKS

In this section, we mention the more relevant and influential works in the literature related to the different aspects of PAVEL design and implementation.

Point sampling. Our basic idea is to use a point sampling approach to place decorations with circular symmetry over the surface of an object. Blue noise point distributions and their variations can undoubtedly be used to roughly pack a surface with circular shapes. Bridson [2007] proposes a simple and efficient algorithm to generate a blue noise distribution. Due to its greedy nature, it tends to leave significant gaps, particularly undesirable for artistic purposes. A more uniform distribution is given by a Centroidal Voronoi Diagram, which can be computed over arbitrary surfaces as described in Xin et al. [2016]. Centroidal Voronoi Tessellation could, in principle, be used also to place non-symmetric elements by using the variant described in Lévy and Liu [2010].

Knöppel and colleagues [2015] introduce a special kind of entity distribution over a surface putting regular striped patterns that can adapt to the local characteristics of the shape, curvature included. This method is useful guidance for scattering points on the surface of the form only onto the stripes. We make use of the latter two methods to place seeds for decorative elements in our pipeline.

Packing. PAVEL can be naturally considered a packing technique. Packing 3D shapes in an arbitrary volume is a well-studied problem, and it is known to be NP-complete. It is relevant in graphics, engineering, operational research, and many other fields. Recent works such as Egeblad et al. [2009], Liu et al. [2015], and Romanova et al. [2018] achieve an excellent volume coverage. Still, they are heavily limited in the orientation of objects, are significantly time-consuming, and can only deal with a limited number of items with few facets. Overcoming these limitations,

Ma and colleagues [2018] propose an algorithmic approach well suited to complex models typically used in graphics. Their algorithm initially scales down all the objects to avoid intersections, then iteratively increases their size, optimizes rotations, and tries some object swapping to arrange local solutions. The algorithm can obtain good results but can take up to an hour to finish the computation to place less than 100 objects. Our approach differs from both, since we pack elements on surfaces, and since we decide element sizes upfront, which makes the problem generally harder.

The packing of 2D shapes is a relevant field of research for visualization and illustration purposes. Reinert and colleagues [2013] proposes an algorithm based on Centroidal Voronoi Diagram relaxation to fill an arbitrary-shaped planar domain with multiple shapes. Saputra and colleagues [2019] also describe an algorithm to pack shapes in a planar domain, with a greater focus on leaving as little space as possible. To achieve their results, they allow deformations of the decorative shapes and add small ones, in the end, to obtain better coverage and reduce space. For similar purposes, Kwan and colleagues [2016] propose a new shape descriptor and use it to measure the touching shapes' quality. Another relevant method, by Chen and colleagues [2017], packs tiles on top of a curved surface for fabrication and decorative purposes. They describe an *attract-and-repulse* algorithm in which tiles are allowed to move and rotate, but not to deform, which can take up to 15 minutes to place more or less 150 tiles. Two works focusing on decorations are Lu et al. [2015] and Chen et al. [2016], which investigate the possibility of transforming surface carving holes following a pattern. In Lu et al. [2015] a texture guides the building of a pattern of connected stripes taking into account the structural problems that can arise when fabricating the final object. In Chen et al. [2016], the pattern generates filigrees using the skeletal representations of simple starting elements. The described pipeline can then place them on a surface, greedily refine their position, and slightly modify them to ensure contact between close elements without overlaps.

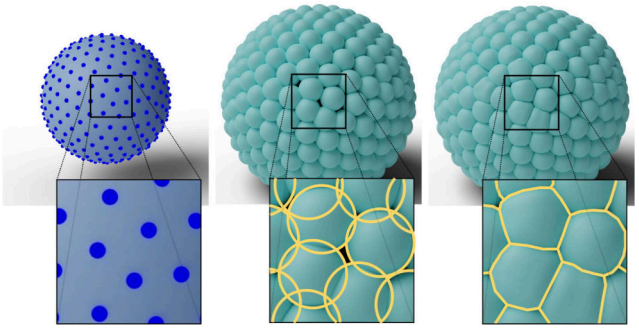
Compared to PAVEL, most of this literature focuses on 2D element placement, and our approach focuses on obtaining the packing effect by deforming the 3D elements keeping the spatial distribution fixed.

Implicit deformable models. Elements' deformation is therefore a fundamental step in our pipeline. Algorithms to control deformable surfaces based on level set formulation are popular in Visual Computing after the seminal works by Osher, Sethian, and Malladi [1995; 1988]. Using implicit formulations, it is possible to model complex interface behaviors, including elastic and plastic deformations, but with huge computational complexity and stability issues [Barton et al. 2011]. A simplified approach is represented by the fast marching algorithm [Sethian 1996], which allows a fast estimation of the evolution of closed surfaces as a function of an underlying velocity field assuming unidirectional front propagation along the surface normals. By controlling the underlying speed, it is possible to drive the surface evolution towards the desired position. Simplified implicit deformable models have been used in Computer Graphics to simulate soft bodies' contact and the resulting deformation [Gascuel 1993] and to animate deformable shapes [Desbrun and Cani 1998]. A similar approach has also been recently exploited in a generalized framework to interactively control the blending of two shapes represented by **signed distance fields (SDF)** on regular grids [Angles et al. 2017]. We adapt the fast marching approach to creating the effect of plastic-like volume-preserving deformation of multiple objects efficiently in our work. A recent work by Fang and colleagues [Fang et al. 2021] proposes a pipeline for the modelling of plastically deformable objects. Their approach combines ideas of geometry processing and physically based modelling, which make it very effective and flexible but, for these reasons, slower than our approach.

Surface extraction. The implicit formulation requires then a subsequent surface extraction to provide an output for rendering and 3D printing. The most common approach to extract a mesh representation from a discrete scalar field is the Marching Cubes algorithm [Lorensen and Cline 1987]. When the scalar field is Boolean, an additional smoothing step is often necessary. Coeurjolly and colleagues [2018] propose an optimization-based approach to extract a smooth manifold from a voxel representation to avoid this. This method works directly on the voxel field's boundary quads, moving the vertices to optimize fidelity, fairness, and smoothness. We do not use this method in our reconstruction, since it is unacceptably slow for resolving our voxel grids. It produces overly smoothed results, which removes details in small decorative elements. We instead use marching cubes and a smoothing remesher to convert the deformed decorations back to a surface.

3D textures. PAVEL is also related to generic methods to generate 3D textures over manifolds. The generation of 3D textures for meshes is an active research topic with many approaches taken for different applications. One of the simplest approaches is shell mapping, introduced in Porumbescu et al. [2005], which generalizes classical texturing to work with thin tetrahedral shells enclosing the manifold. This allows the creation of fully 3D textures but also shares the classical texturing drawbacks, such as warping, deformations, and stitching. To reduce these issues, Zhou et al. [2006] introduce a low-distortion parametrization of shell space and a matching step to improve the texture tiling. A different approach is to work directly on the surface, as described by Fleischer et al. [1995], thus avoiding the parametrization problems. This method is based on a simulation that places spheres over the manifold and replaces them with the desired shape after the simulation has ended, offering no guarantees on mesh intersections, which are indeed bound to happen when replacing spheres with anisotropic shapes. Bhat et al. [2004] and Ma et al. [2011a] describe two different approaches for 3D texturing by example, focusing, respectively, on surface and volume textures. While the results are remarkable, and in the case of Bhat et al. [2004] even close to our goal, these methods are limited by the fact that an example, which would have to be sculpted manually, the pattern structure is not controllable, and the examples' shapes are drastically deformed. While it may be possible to adapt some of these works for our case, especially Zhou et al. [2006] and Fleischer et al. [1995], they are all significantly slower than point sampling and none of them can handle soft body deformations.

An important aspect in the texture generation with PAVEL that is not found in other methods is that the element-based approach



(1) Point sampling: manual, isotropic, striped

(2) Initialization: select, combine, scale, rotate

(3) Disentangling and deformation: volume recovery function

Fig. 3. The proposed pipeline of *PAVEL*. Seeds can be placed on the base manifold manually, isotropically, or in striped or gridded patterns to create an initial configuration of elements over the base shape with partial overlap between the elements. Elements' rotation, scaling, and types can be changed with specific options. The deformation step creates the final shape simulating packing of plastic, volume preserving elements, with some control on the material behavior obtained with the volume recovery function. The final result can be exported as a novel shape or as a collection of printable shell elements.

allows an exact characterization of textures with quantitative attributes related to the elements' distribution (density, orientations, grid, etc.) that can be used to control the pattern design.

3 OVERVIEW

Our goal is to easily create element-based relief patterns over arbitrary surfaces. Our algorithm takes as input a base mesh, a mesh for each decorative element, and a small set of parameters, defined in the later sections, that control the elements' distribution and deformations. Our method, shown in Figure 3, works in two stages. We first generate a point distribution over the base surface such that the decorative elements cover the surface well. In the second step, elements are placed in the location previously defined, and our algorithm computes the elements' deformations to eliminate the overlaps among them.

To achieve realism and mimic real-world objects, we position the decorative elements so they are partially overlapping, and then transform the elements' shapes. They are non-overlapping at the end while preserving their volume. This approach simulates the plastic deformations that artisans perform on their creations. While, at first sight, using a physical simulator seems the proper method to address the second step, we have found that they do not work in our case. The first concern is that simulators are very slow at our resolution and number of collisions. More importantly, though, the simulation does not necessarily converge to a meaningful state, since we start with significant overlaps with a non-physical nature. Instead, we propose a new method based on an implicit representation of the elements and the **Fast Marching Method (FMM)** [Sethian 1996], which can generate physically plausible plastic deformations with bulges near the contact surfaces. We discuss this algorithm in Section 4.

We propose two strategies for the automatic placement phase that enable users to create both isotropic and anisotropic distributions with simple control parameters. These are, respectively, based on the constrained Voronoi algorithm proposed by Xin et al. [2016] and the equidistant stripe patterns from Knöppel et al. [2015]. In case artists want direct control over the decoration placements, they may manually do it. Additionally, in the case of base surfaces with high-curvature regions and decorations with dimensions in the same order of magnitude of the base curvature radius, we allow the user to sample an isosurface at offset o to produce a more even distribution. We then use these points to center the decorations on them and produce intersections among adjacent elements. This step, and all its parameters, are discussed in Section 5.

As the pipeline aims to generate object design both for rendering and manufacturing purposes, we discuss in Section 6 how to extract smooth meshes from the voxelized elements' sets and how to generate printable shells for the decoration of real objects. A gallery of examples of decorations obtained with *PAVEL* is shown in Section 7 with an analysis of the system performance, while Section 9 evaluates limitations and possible improvements of the approach.

4 DEFORMATION OF PACKED DECORATIONS

In this section, we describe how to compute the final deformations. Our algorithm takes as input a set of *overlapping* element decorations placed onto an object's surface. Our goal is to remove the overlap by computing each decorative element's deformation while maintaining its volume to simulate the plastic deformations performed in real-life by artists.

To achieve this goal, we propose a novel method based on an implicit formulation, inspired by works on interactive visual simulation of volumetric object contact [Gascuel 1993]. We rely on fast marching [Sethian 1996] using a specifically designed parametric function for the propagation field that determines the deformed shape appearance and enforce volume preservation. This allows us to obtain a final result that is close to the one obtained in the real artistic procedure without the need to perform a complex physical simulation. This simulation would require initializing objects without intersections, controlling the forces used by artists to place and deform the elements, and simulating the plastic behavior accurately. As far as we know, it is not possible to obtain an accurate simulation of the whole artistic procedure, even if it is possible to create procedures aimed at creating automatic deformations of multiple elements. As shown in Section 8, it is, however, difficult to tune the large number of simulators' parameters; there are stability issues and the computational time is significantly higher than ours.

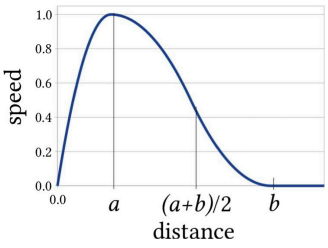
We voxelize each decoration d_i in its own dense grid g_i that is 50% larger than the decoration's dimension to ensure that the computed deformation will fit in the grid. Compared to the whole surface, this is a sparse discretization that keeps memory consumption relatively low, since we are effectively discretizing a slab around the surface. While each decoration has its own representation, we ensure that the voxel dimension and the grid rotation are consistent across all voxelizations. As a consequence, we can perform Boolean operations just by keeping into account their mutual offsets. Additionally, separate voxelizations, instead of one global

labeled grid, allow for simple handling of the overlaps and compatibility with the fast marching algorithm.

Given as input a set of voxelized decorations $D = \{d_0, d_1, \ldots, d_n\}$, our first step is to resolve the overlaps. We do this by detecting the voxels shared by multiple decorations and removing these voxels from all the decorations but the one with the shortest distance from the decorative element center. Furthermore, we remove all the voxels of the decorations shared with the base objects as well. These removals generate a new set of decorations D', where each decoration has associated a measured volume loss v_i, equal to the sum of the voxels removed in the previous steps. We then ensure that each element recovers its original volume by expanding it in a way that simulates a plastic deformation.

Fast marching approach. To compute the expansion, we use a fast marching approach [Sethian 1996] to grow each element in the empty regions until they recover the original volume. To simulate a plastic deformation, which typically generates bulges near the soft bodies' contact points, we use a non-constant velocity field to drive the fast marching algorithm. For each shape S_n, we find the contact surfaces with the other shapes, and from them, we build the respective distance fields. The distance fields are then converted in velocity fields F_n through a parametric formula, the *volume recovery function*, which can be adapted to obtain the desired behavior. The recovery function is a piece-wise polynomial, obtained by connecting three second-degree polynomials and enforcing C^1 continuity and defined by:

$$f_{a,b}(x) = \begin{cases} 0 & x \leq 0 \\ -\frac{x^2}{a^2} + 2\frac{x}{a} & x > 0, x < a \\ -\frac{2(x-a)^2}{(b-a)^2} - 1 & x > a, x \leq \frac{a+b}{2} \\ \frac{2(x-a)^2}{(b-a)^2} + b(x-a) + 2 & x > \frac{a+b}{2}, x \leq b \\ 0 & x > b. \end{cases} \quad (1)$$



In the inset, we show an example plot of the function. We keep the velocity null in contact points to simulate the constraints related to the attachment of surfaces. We then have two control points a, b used to define the distance for maximal bulging and the maximal distance reached by the plastic wave. This allows effective control of "material behavior" driven by the amount and locality of the deformation.

The velocity fields are then weighted proportionally to the volume losses to simulate a faster growth of the elements more "squeezed" by the neighboring ones and used to compute the new arrival times for each decoration. To recover the volume lost by each decoration, we evolve the front of each decoration with the fast marching method. For each elements' sub-grid g_n, a set of active front voxels is initialized, taking those adjacent to the initial voxelization border. In contrast, we set all voxels belonging to either the initial shape, the base shape, or other elements to inactive. Each decoration can recover its volume only by occupying empty voxels.

The front propagation is estimated based on the solution of the Eikonal equation:

$$|\nabla T| = F, \quad (2)$$

linking the velocity function F to the arrival time of the front T. Given the discretization of the spatial derivatives

$$\begin{aligned} D^{+x}T &= T(i+1,j,k) - T(i,j,k), \\ D^{-x}T &= T(i,j,k) - T(i-1,j,k), \\ D^{+y}T &= T(i+1,j,k) - T(i,j,k), \\ D^{-y}T &= T(i,j,k) - T(i,j-1,k), \\ D^{+z}T &= T(i,j,k+1) - T(i,j,k), \\ D^{-z}T &= T(i,j,k) - T(i,j,k-1), \end{aligned} \quad (3)$$

the first-order solution for the arrival times is obtained by estimating the arrival time for all the active front voxels $T(i,j,k)$ with the discretized Eikonal equation:

$$\begin{aligned} [max(D^{-x}(i,j,k)T(i,j,k), D^{+x}T(i,j,k), 0) + \\ max(D^{-y}(i,j,k)T(i,j,k), D^{+y}T(i,j,k), 0) + \\ max(D^{-z}(i,j,k)T(i,j,k), D^{+z}T(i,j,k), 0)]^{\frac{1}{2}} = \frac{1}{F_{ijk}}, \end{aligned} \quad (4)$$

where

$$F_{ijk} = \sum_c \omega_c f_{a,b}(r/d)$$

is the space-varying speed function depending on the distances r_c from the initial contact surfaces with the neighboring volumes, evaluated at each voxel location, and ω_c is a weight proportional to the volume lost in the intersection. The scaling parameter d is computed as the cubic root of the element volume.

The algorithm evaluates the arrival time for all the voxels of the active front. It selects the voxel with a minimal arrival time according to the Eikonal equation's solution estimated on the base of the velocity field defined in the corresponding grid. The selected voxel is then assigned to the element, and the active front is finally updated, removing the selected voxel from it and adding to it the voxel's free neighbors. The procedure is repeated until all the original volume of the element is recovered.

It is theoretically possible that the algorithm could not find enough voxels to recover the volume when the evolution of neighboring elements occludes the front, but this is not likely to happen for a layer of small decorations over a base surface and never happened in our experiments.

While fast marching can be parallelized [Yang and Stern 2017], it comes with additional complexity and overhead. Instead, we use the classic serial implementation that we run concurrently on multiple decorations. To avoid race conditions, we precompute the fast marching in parallel for each decoration independently, and then we sequentially assign the voxel using the precomputed arrival times. Collisions between evolving fronts are handled by setting the voxels reached by one front as not usable by the other elements.

The method ensures by construction that the volume is correctly maintained while deforming the decorations.

Our implementation is based on the fast marching functions of Scikit FMM [2020] and handles IO and voxelization with Trimesh [2020].

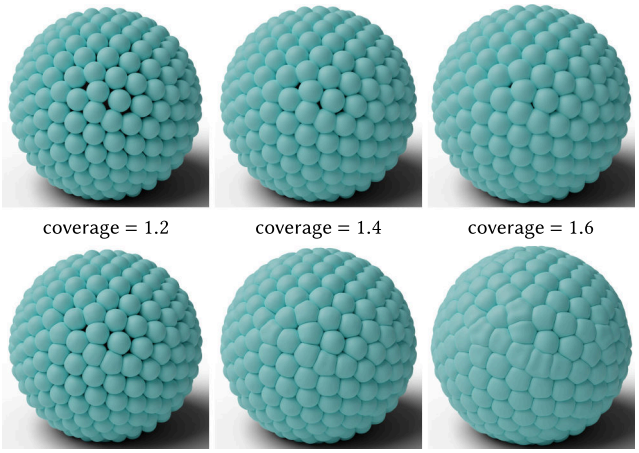


Fig. 4. The effect of different values for surface coverage on the final decoration. *Top:* initial placement. *Bottom:* packed elements.

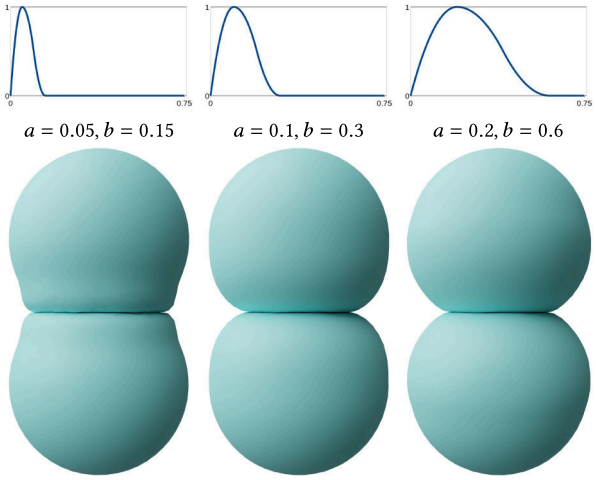


Fig. 5. The effect of different choices for the volume recovery function parameters on the deformation of a couple of spheres isolated from Figure 4.

Surface coverage control. The method described above can simulate the elements' plastic deformation if the decorative elements are initialized with a reasonable overlap. To understand what is a "reasonable" overlap, we performed several tests to define control parameters for the point distribution. To control the decorations' overlap, we choose to use the *surface coverage*, that is, the ratio between the sum of the surface areas of the decorations' footprints and the base object area to be decorated. Figure 4 show results of packing small spheres on a base sphere with different surface coverage values. As we can see, a ratio of 1.2 is barely enough to cause some deformation, while a ratio of 1.6 produces excessive and unrealistic distortions of the elements' shape.

Volume recovery parameters. We also performed extensive tests to choose the parameters of the volume recovery function (Equation (1)). Figure 5 shows the results obtained with the same overlapping spheres but different choices for the values of the a and b parameters. By shifting the control parameters, it is possible to simulate different plastic behaviors. All the results shown in the article, but Figure 5, are obtained with the same settings ($a = 0.1, b = 0.3$).

As it is possible to see, for example, in Figure 6, this choice provides results that are visually quite close to real plastic deformation.

5 PLACEMENT OF DECORATIONS

The deformation step is enough to obtain the final result if the user places the decorations manually on the base object's surface, as shown in Figure 6. In this manner, the user directly controls the decorations' location and the surface coverage, while *PAVEL* controls the material behavior using the deformation parameters described above. In the system, we supply the automatic placement methods described in the following.

Automatic placement by Point Sampling. While manual placement provides complete artistic control, the decoration of complex surfaces with small decorative elements would be very time-consuming, just like in real-world crafts. To simplify creating complex decorations, we also support the automatic placement of decorative elements using two procedural methods.

Our placement methods are based on point sampling algorithms. We support the placement of objects with approximate 2D radial symmetry, with the symmetry axis aligned to the base surface's local normal. We also assume that the decorations are small compared to the base surface's curvature to ignore the variations in the local volume extruded from the base surface. Both of these are reasonable assumptions found in the hand-crafted models that inspired our work.

Instead of a packing algorithm, our use of point sampling is motivated by a tradeoff between speed and packing quality. It is worth reminding that optimal packing is NP-hard in the Euclidean domain and just as hard in the intrinsic manifold metric. This means that all published packing algorithms are approximations that trade off element shapes, packing quality, and computation time. In general, it is well known in both the literature and the practice that most packing algorithms fail or are not optimal for all but the simplest shapes. Note also that we do not allow to change the decoration size, which is the most used approximation for soft packing in graphics. For all these reasons, our point sampling approaches are just another heuristic within their stated constraints. If better placement heuristics are available, then we could integrate them into our pipeline with ease.

Isotropic seeding. Our first automatic placement method produces isotropic distributions that work well for decorations with cylindrical symmetry. The user controls the density of decorations by setting the desired surface coverage, and the algorithm instantiates the correct number of seeds to guarantee the target density.

To ensure a high-quality distribution, we rely on optimization using a method based on the **Centroidal Voronoi Tessellation (CVT)** and its efficient implementation described in Yan et al. [2009] and available in the Geogram library [Levy 2020]. We generate the base surface's tessellation with the desired number of vertexes and use them as the centers of the decorations. When

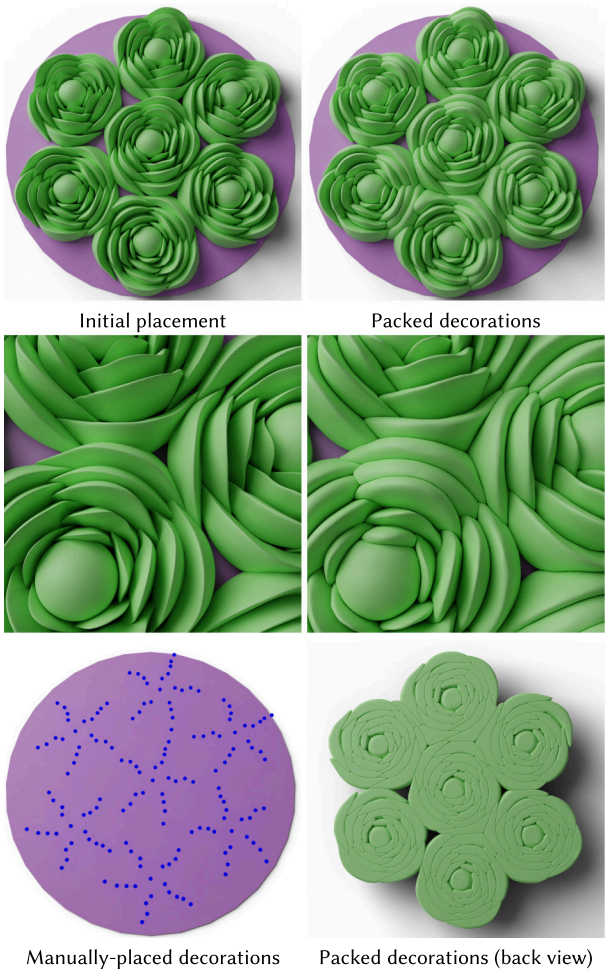


Fig. 6. Example of decoration obtained by manually placing elements and having *PAVEL* resolve the overlap. Here each petal is a separate decorative element and the deformation is computed to simulate sculpting clay. Note from the back view how *PAVEL* can resolve the overlap in a very complex case with many self-intersecting, non-symmetric, elements.

converged, this iterative approach produces a point distribution with good statistical properties, resulting in a high-quality decorated surface. To obtain a more "hand-made" look, we allow users to reduce the number of iterations. Figure 7 shows results obtained with this method that closely resembles the hand-made objects shown in Figure 2.

Offset surface. The previous method works well if the decorative elements are small compared to the surface curvature. If that assumption is not valid, then the decorated surface has visible artifacts, shown in Figure 8. The main problem is that our method generates equally spaced seeds that represent the bottom of the decorations but does not take into account decorations' volume and surface curvature. Decorations overlap more in concave regions than convex ones. The volume above the surface is smaller in the former case and can result in heavily compressed areas in concave zones and non-contacting in convex regions. When

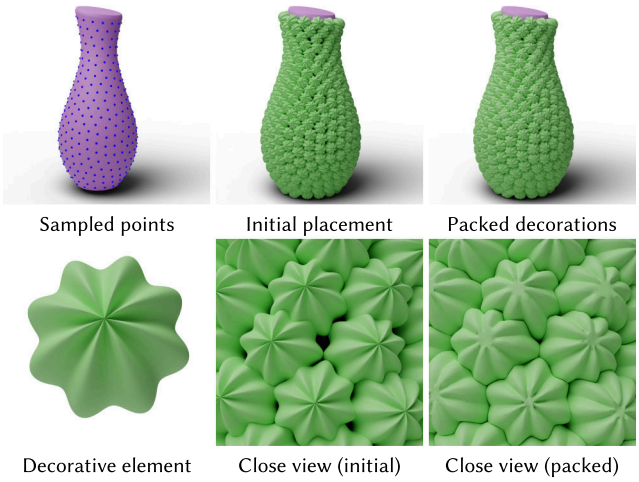


Fig. 7. Isotropic seeding with CVT on a vase.

decorations are small compared to the surface curvature, this effect is negligible. But it becomes more noticeable as the decorations' size increases.

For this reason, we also provide a method that considers the volume of the elements when point sampling their locations. The basic idea is to extract an isosurface at a signed distance d from the base mesh, perform the sampling on it, and then map back the seed points on the original base surface, projecting them on the local normal direction.

An appropriate choice of d is the height of the element's z-section with the maximal area. In this way, we can handle large decoration sizes even on surfaces with varying curvature. We show a clear example of this in Figure 8. Here, the bunny features gulfs and bulges. They are handled without problems by sampling the offset surface. Conversely, sampling on the surface reveals significant artifacts. Since decorations are small in general, and the modified sampling has a non-negligible computational cost (see Section 7), we leave the choice of whether to use the offset surface to the user.

Stripe-based, anisotropic seeding. To support the automatic generation of anisotropic element patterns, we also provide a stripe-based sampling approach. This method can create gridded patterns of decorations with approximate cylindrical symmetry and even with elongated shapes.

Our method is based on Knöppel et al. [2015], which allows us to generate patterns of evenly spaced stripes over a generic base surface. The method is based on a simple optimization problem and approximately maintains a user-specified orientation and spacing of the lines by automatically inserting branch points where necessary. Furthermore, the generated patterns are globally continuous, which avoids visible seams after the applications of our decorative elements and with frequencies not depending on the mesh resolution. We exploit the stripe-based method by generating two crossing stripe patterns with a user-defined crossing angle and sampling seed points at the line crossings. In this manner, we obtain lines with points sampled at constant distances and with a controlled offset among the decorative elements of two adjacent stripes.

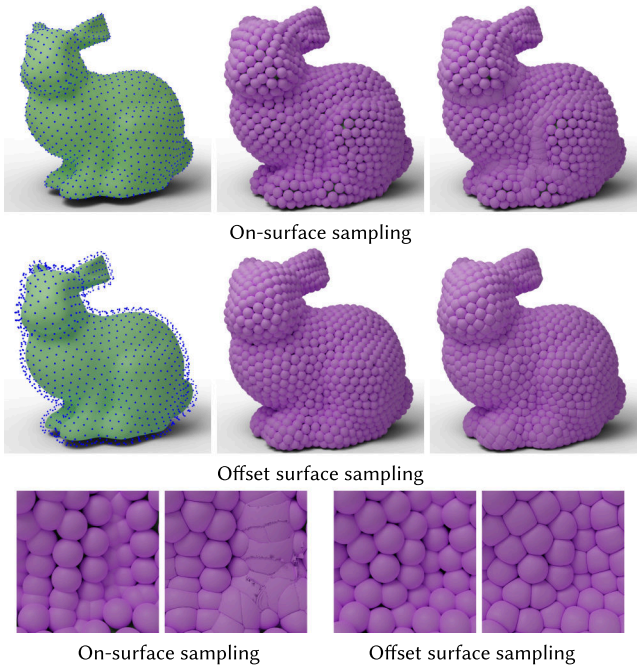


Fig. 8. Comparison between sampling on the surface (top) or on an isosurface at distance d (middle). Using the isosurface allows for more even packing and deformations.

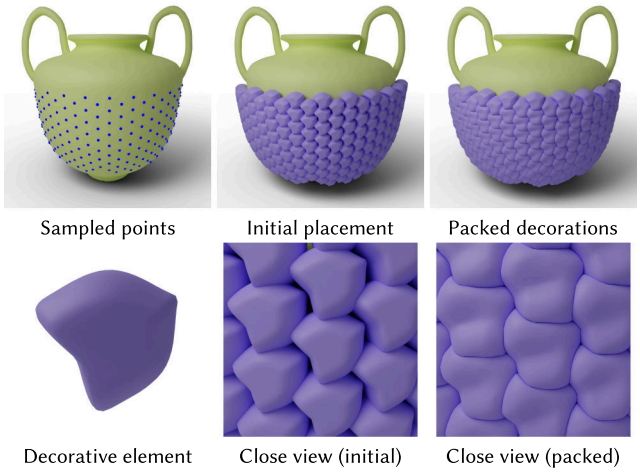


Fig. 9. Non-isotropic elements' sampling on non-perpendicular stripes' directions.

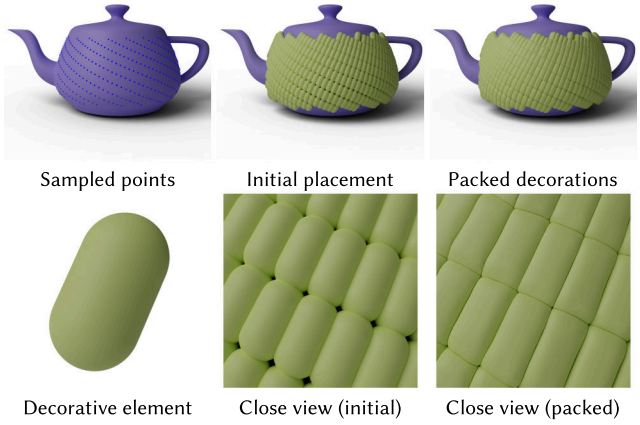


Fig. 10. Using the stripe-based sampling, we can pack on a generic shape like the teapot non-isotropic decorations aligned with a directional field.

Figure 9 shows an example with crossing stripes creating an angle of 60 degrees and elements placed with a constant orientation concerning the main stripe.

Since we can independently set the sample spacing on the main stripes and the distance between adjacent stripes, we are no longer forced to pack objects with approximate radial symmetry, but we can insert elements with arbitrary elongation. Furthermore, since the stripes encode a local directional field, we can use non-isotropic decorative elements oriented in a controlled way with respect to the stripes' directions. Figure 10 shows an example with perpendicular stripes-based sampling and non-isotropic elements with main directions aligned with the stripes fields.

In our stripe-based algorithm, we create the sampling points and estimate and store the point connectivity of the sample along with the two stripe directions. In this way, we can develop decorations alternating different elements and elements' orientations, as shown in Figure 11.

Elements' perturbation. Aside from placement, we can easily integrate controls for decorations' rotations and filling different areas with different decorations. The rotation for each element can be set randomly or aligned to a field defined over the object surface, giving the possibility to create different artistic effects that work particularly well when placing elongated objects over a gridded seed distribution. Figure 11 shows an example of an artistic pattern obtained with alternate rotations of 180 degrees of the elements along the principal stripe field direction.

The average elements' size is chosen to guarantee the desired decorative effect, and the number of elements can be tuned to guarantee the desired amount of initial overlap. However, we leave the possibility of applying a further random perturbation to the size of each decoration to increase the number of design options. We allow variations up to ±10%, as larger ones would lead to inconsistent overlaps and too many empty spaces.

Last, it is possible to load more than one decoration and randomly select one of those for each seed. We can exploit this to increase the final results' variety loading a multitude of slightly different decorations (e.g., stones of various types, flowers).

Figure 12 shows an example of decoration mimicking the one in Figure 2 obtained by using three different elements shapes and random rotations with respect to the main field direction.

6 OUTPUT GENERATION

The last step we need to perform is to convert the process results into high-quality meshes for visualization or printable shells to decorate real-world objects.

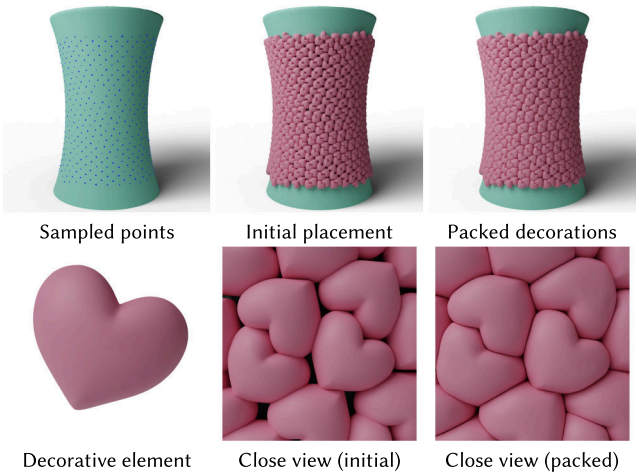


Fig. 11. Example of alternate elements' orientation along the main field direction.

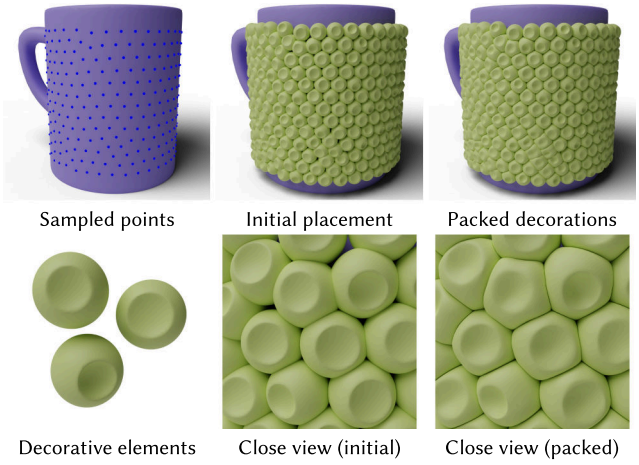


Fig. 12. Using multiple elements' shapes and random orientations it is possible to create a variety of artistic designs.

High-quality meshes. To generate a 3D mesh of the whole object, we first independently create meshes for each deformed decorative element using the Marching Cubes algorithm [Lorensen and Cline 1987]. Then, we merge the elements and the base object into a single model and remesh the result using the Central Voronoi Tessellation approach [Yan et al. 2009], which provides feature-preserving results and is robust and efficient for large meshes.

Printable shells. It is possible to use our pipeline also to create decorative shells for real-world objects. To decorate an object, we first scan it to obtain an accurate digital model and then create a *PAVEL* decoration for it. This produces a mesh representing all the decorative elements fused together. We then decompose this final mesh into printable parts and wrap it around the original object. Figure 13 shows the whole process of decorating an object.

For objects with a simple topology, the decomposition works as follows: First, we segment the base object. Any mesh partitioning can be used for this purpose. In the example shown, we cut the object with two mutually perpendicular planes passing through the model's symmetry axis. Given the base segmentation, we cluster the deformed elements placed on each patch and remesh them into single shell elements to be 3D separately printed. One can use a more appropriate decomposition for more complex base objects while the rest of the pipeline remains the same.

We obtained the final tangible result shown in Figure 13 by gluing the shell parts created from the design over the base object. In this case, the object was a mug for which we produced the corresponding digital model, scanning it with a LMI Gocator 3210 scanner. We decorated it with an isotropic distribution of elements with different shapes and randomized orientation, and we finally printed the shell elements with a DWS XPRO S 3D printer.

7 RESULTS

Reproducing real-world decorations. Using *PAVEL*, we can create a variety of decorations that mimic real-world artworks. Figure 14 shows side-by-side views of original artworks, courtesy of Heather Knight, and objects generated with our pipeline and rendered with clay-like material properties.

At the same time *PAVEL* is a tool applicable to any base form, from the simplest (like the sphere in Figure 15) to the more varied in curvature (like the cactus in Figure 15 or the fertility statuette in Figure 1). But *PAVEL* gives its best in designing artistic decorations, which are not limited to the replicas of clay pieces but can be used to generate various artworks of different kinds. Figures 16 and 17 show a few examples, demonstrating potential uses in jewelry and pottery, exploiting manual elements positioning and automatic pattern creation.

To evaluate the advantages of the computer-assisted procedure with respect to a manual artistic one, we performed a simple user test asking a designer trained to use the PAVEL framework to reproduce two simple artworks of Heather Knight, the vase and the tile shown in Figure 14, top left. The time required was limited to a few minutes for the choices of the options and the element, with a subsequent overall computing time for the automatic placement and elements' deformation of about 1 minute (see Tables 1 and 2). According to the artist, the time required to create the clay artworks is about 2 hours for the tile and 8 hours for the vase.

Validation. Since *PAVEL* does not perform a real physical simulation, there is no ground truth to compare with. Instead, since the goal of our work is to design patterns resembling hand-made artistic creations, we designed and conducted a user study aimed at understanding if the generated patterns mimic real artworks well.

In the user study, the subjects had to choose for each image in a set, whether they were photos of actual artwork or *PAVEL* generated models. Figure 18 shows the test images that include cropped photos of artworks by Heather Knight (H1–H4), rendered images of *PAVEL* decorations (P1–P3), or photos of a printed *PAVEL* shape (P4). Sixty-two subjects participated in the experiment. Twenty-two had no specific background, 32 declared to be experts in computer science, and 8 claimed to be experts in computer science and digital design. Subjects had to compile a web form with the photos presented with a corresponding radio button to select the desired option.

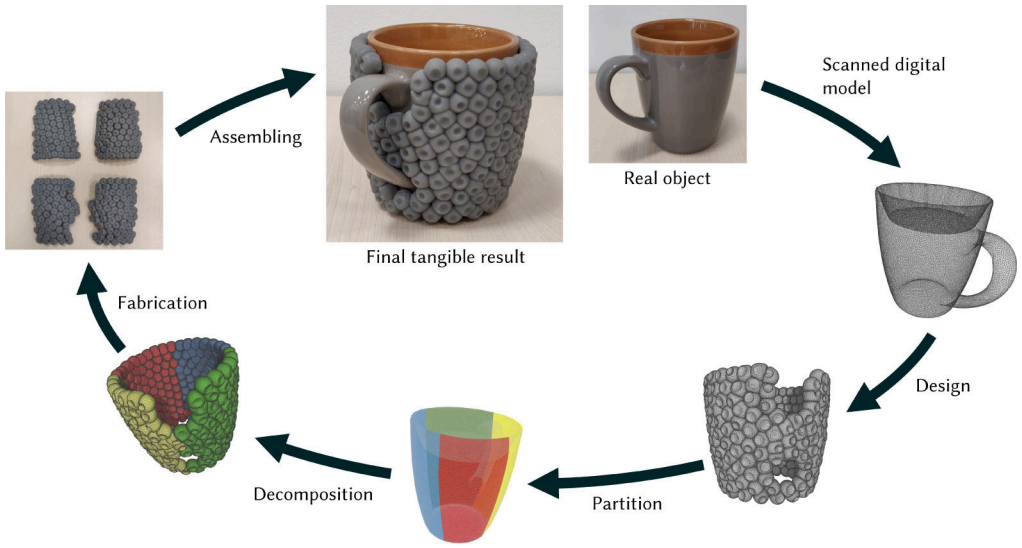


Fig. 13. Design and fabrication of tangible decorative shell for a real object. The whole production pipeline is depicted: From the real object, we want to decorate to the final tangible result obtained by applying the fabricated decomposed shell over the surface.

We summarize the results of the experiment in Figure 19. The percentage of subjects considering the pattern hand-crafted is higher than 30% for all the patterns generated with *PAVEL*. This is a very good result considering that even though the question asked explicitly to consider the geometric pattern and not the material reflectance and that we added some noise to the synthetic rendering, there was probably a bias penalizing the renderings. This is confirmed by the fact that the only photo of a printed *PAVEL* pattern (P4) was believed hand-crafted by more than half the subjects. Moreover, the percentage is significantly higher than those of actual hand-crafted patterns (H3, p-value in 1-tailed 2-populations z-test 0.014). It is also worth noting that statistical tests also show that the proportions of *hand-crafted* votes for the two rendered *PAVEL*-generated patterns P2, P3 are not significantly different 2-populations z-test from those of the real artworks H3, H4. P2 and P4 were also considered *handcrafted* by more than half of the subjects hat declared themselves experts in Computer Science and Digital Design.

We are also proud to report what Heather Knight, the artist that inspired the methods we developed, commented when we showed her some renderings of the generated artifacts. She was blown away and appreciated much the possibility offered by our system.

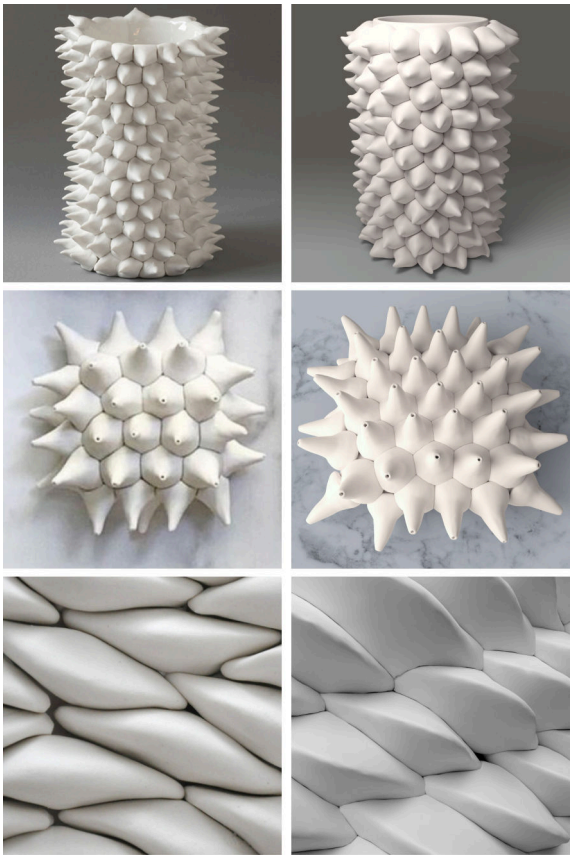
Performance. To understand if *PAVEL* could be a useful design tool, it is necessary to evaluate the time required by the different steps and the global effort required to create the decorated models. We obtain all our results on a 12-core Ryzen 3900x with 32 GB of RAM. Table 1 shows the time required by the automatic seeding procedure. All the times for the non-curvature adaptive methods range between 0.6 and 1.5 seconds, with minimal differences between CVT and striped seeding. The most computationally expensive sampling strategy is, as expected, the curvature-dependent one. We used this method to sample the fertility and the bunny model, respectively, shown in Figures 1 and 8, which took between 30 and 60 seconds to extract the isosurface and only a couple of seconds to sample it. This speed allows a reasonably fast interactive design procedure. The digital artist can quickly test different object choices and seeding options visualizing the seeded configurations superimposed on the base model.

Table 2 shows the times required by the volume recovery and mesh generation. While the whole procedure is clearly not interactive, the time required is reasonable for an offline rendering procedure. The time required by the volume recovery step is directly dependent on both the number of decorations and their resolution. The times range from 20 to 360 seconds for all but one mesh, with the faster times achieved by the models with around 50 decorative elements. The slowest time is reached by fertility, which stays just under 10 minutes, because of its 2,000 decorations and the 130 millions of voxels used to represent them. Another critical factor in performances is the resolution, which is also crucial to maintaining the decoration's details. Our results show how the method can handle millions of voxels to represent the decorations, even in intersections up to 33% of the volume. Last, another time-consuming step is CVT smoothing. While it ranges from 10 to 200 seconds to remesh most of our test models, it is worth noticing that the marching cubes meshes' resolution reaches up to 60 million triangles, and we output smooth meshes with up to 5 million triangles.

8 COMPARISON TO PHYSICAL SIMULATORS

The use of existing commercial or publicly available physics simulators might appear to be the obvious choice to produce results similar to our own. In fact, we initially explored three different alternatives using them, but with no practical results.

The first possibility would be to simulate exactly what artists do to create decorations. In the real world, decorations are placed one-by-one and deformed during placement to match the overall design constructed so far. To do this, we would need to be able to simulate the material accurately and generate the exact forces that artists apply manually, which are different for each decoration. While an



Real-world objects *PAVEL* replica

Fig. 14. Comparison of element-based clay hand-crafts created by an artist (left column, courtesy of Heather Knight) and the corresponding rendered models created by *PAVEL* using automatic seeds placement (right column).

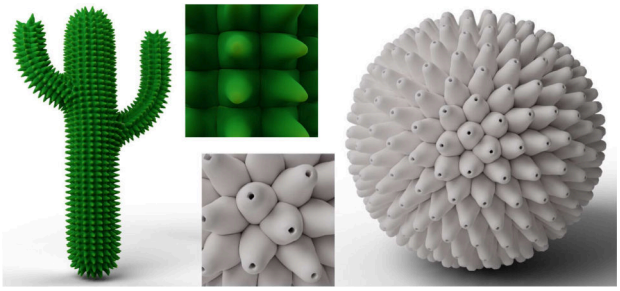


Fig. 15. We show here how *PAVEL* can deal with both regular base shapes (the sphere on the right) and shapes with sharp curvature changes and creases (the cactus on the left).

accurate material simulation is possible, the control with the exact forces applied by artists is not easy to formalize algorithmically.

As an alternative, we can attempt to use physics simulators to produce results similar to our own, but without respecting real-world physics. Essentially, we can attempt to use physics simulators as black-box algorithms that solve specific computational problems, such as collision avoidance and constraint solving.

Fig. 16. Some jewels designed with *PAVEL*. On top: a decoration made of seven pink gold roses with a pearl in the middle; we use two different metal polishing for the final rendering. On the bottom part, there are two rings. We show details of the decorations and, for one of the rose decorations and the golden ring, also the packing of the decorations on the back, over the surface, visible removing the base shape.

Fig. 17. Two vases designed with *PAVEL*. For the jade one, we show also the back of one of the parts that could be used for fabrication.

The simplest possibility is to set up a configuration like our own, namely, place a set of overlapping objects and ask the physics simulator to resolve the overlap and generate a final configuration that is overlap-free and looks like plastic deformation. In this case, the simulators start from a non-physical configuration that has significantly large overlap to produce packed decorations. We tested several simulators, namely, Houdini's FEM solver [SideFX 2020], Houdini's XPDB solver [Bender et al. 2014; Macklin et al. 2016], Blender's solver [Blender 2020], and a projective dynamics solver

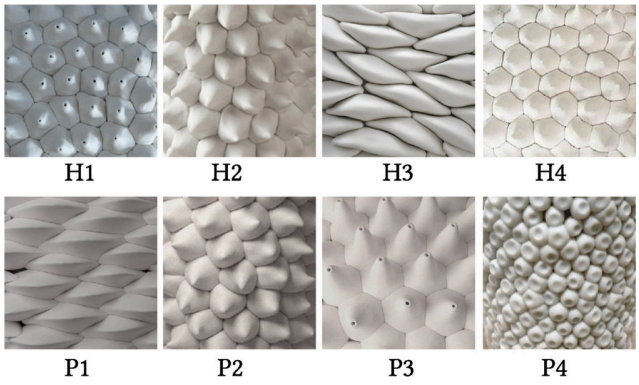


Fig. 18. Images proposed to the subjects in our user study. H1–H4 are cropped photos of real artworks (Courtesy of Heather Knight). P1–P3 are renderings of patterns generated with *PAVEL*, while P4 is a cropped photo of a 3D print of a pattern generated with *PAVEL*.

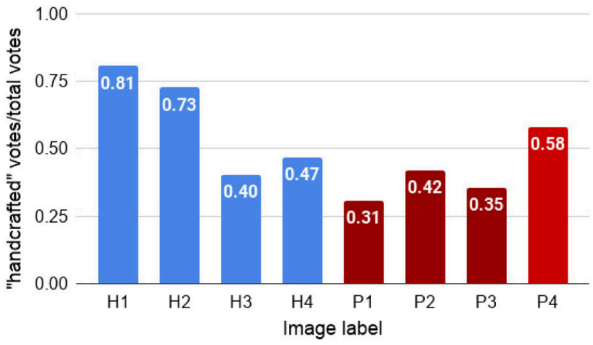


Fig. 19. Percentages of subjects that considered the images proposed in the user study representing handcrafted patterns.

[Fratarcangeli et al. 2016]. In all cases, we either could not remove the overlap or converged to a configuration with significant artifacts on all decorations. The main reason for this is that the starting configuration has too much overlap, which is non-physical, while simulators are built to evolve from a valid configuration to another.

The last alternative we tested is to start the simulations in a physically correct configuration by scaling down all decorations around their center until there are no overlaps. We then inflate the decorations to recover their volume, while the simulator ensures that no overlap occurs by applying collision detection and response. If the inflation is performed unconstrained, then the original decoration shape is entirely lost. Instead, we apply constraints to maintain the decoration shapes approximately. We perform this test using only the Houdini XPDB solver [Bender et al. 2014; Macklin et al. 2016], since a position-based dynamics formulation makes it easy to combine constraints to obtain specific effects, and the particular implementation we chose has flexibility in constraint manipulation, good numerical stability, and it is used extensively in the entertainment industry to guide simulations for artists effects. Note once again that we are combining constraints in a non-physical manner, since we want the decoration to inflate like air balloons while converging in a configuration similar to clay.

In our XPDB setup, we use a shell-based simulation with three constraints to control the inflatable object, in addition to the collision constraints to avoid overlap. We combine an edge preserving constraint, a bending constraint and a pressure constraint, described in Bender et al. [2014], as suggested by the solver vendor. We set up the solver and constraint parameters by running several instances of the experiment and pick the best results. We also tested shape- and volume-preserving constraints, which would seem to better preserve shape, and obtained worst results, since they interact badly with collision-handling constraints.

Table 1. Sampling Parameters for the Results Obtained Via Automatic Decoration Placement

Model	Number of decorations	Surface coverage	Time
Amphora, Figure 9	358	1.42	0.59 s
Bunny (adaptive), Figure 8 middle	1,720	1.23	2.68 s
Cactus, Figure 15 left	1,244	1.18	0.42 s
Fertility, Figure 1 left	2,002	1.23	1.8 s
Mug, Figure 12	410	1.42	0.68 s
Mug, Figure 13	297	1.44	0.92 s
Ring, Figure 1 middle	529	1.30	1.00 s
Sphere, Figure 15 right	400	1.43	0.69 s
Teapot, Figure 10	642	1.26	1.82 s
Tile, Figure 14 middle	48	1.49	0.80 s
Tile, Figure 14 bottom	65	1.32	1.30 s
Vase, Figure 7	450	1.49	0.70 s
Vase, Figure 11	953	1.27	0.91 s
Vase, Figure 14 top	182	1.48	1.43 s
Vase, Figure 17 right	604	1.38	0.79 s
Vase, Figure 17 left	555	1.44	0.77 s

The reported times for fertility and the bunny are not inclusive of the isosurface generation, respectively, 33.7 s and 54.2 s.

Figure 20 shows a comparison between an element-based decoration generated with PAVEL and the corresponding best result we obtained by inflating non-intersecting elements with the simulator, adding the shape constraints. We match the resolutions of the two results by remeshing the edges of the simulator's triangles to have the same edge lengths of PAVEL's voxel edge length. The simulator is significantly slower than PAVEL: Here, the execution time is 5 hours and 56 minutes while our system takes only 8 minutes. We also believe that our results are qualitatively superior, since the packing is more pronounced, resulting in a better coverage of the surface.

The superior quality of PAVEL results is even more evident in Figure 21, showing another example of decorative elements' deformation obtained with the previously described simulator setting, compared with PAVEL results and also with a similar real-world object. Here the simulator takes 2 hours and 14 minutes for the computation (PAVEL time is 1.5 minutes). The closeup in Figure 21 shows that our result better matches the behavior of real clay. In Table 3, we report the speedup provided by our method, as well with the number of voxels and triangles handled by, respectively, *PAVEL* and the physics engine.

As with all simulators, it is possible that a different setup based on XPDB may obtain better results, for example using tetrahedral meshes with volumetric constraints. However, the computational complexity would be even higher, making these setups not competitive with our system for our specific task.

Table 2. We Report the Execution Times (in Seconds) for the Two Most Expensive Pipeline Steps, Deformation and Output Generation, for the Set of Models We Discuss

Model	Deformation time	Marching cubes and Smoothing	Voxelization avg. diagonal	Decorations voxels	Overlapping voxels	Base voxels	Marching Cubes triangles	Final triangles
Amphora, Figure 9	112.5 s	58.6s	154.5	31.68M	30.2%	64.71M	12340K	716K
Bunny (adaptive), Figure 8 middle	357.8s	138.6 s	123.9	96.24M	13.7%	204.00M	38487K	1745K
Cactus, Figure 15 left	239.3 s	185.2 s	139.8	50.37M	22.3%	78.31M	25714K	2856K
Fertility, Figure 1 left	574.4 s	409.7 s	140.0	130.74M	13.9%	154.60M	57909K	4890K
Mask, Figure 1 right	322.1 s	74.7 s	150.9	32.26M	16.6%	13.64M	8518K	472K
Mug, Figure 12	121.7 s	59.8 s	154.3	43.23M	17.0%	200.23M	14305K	850K
Mug, Figure 13	108.9 s	47.2 s	170.4	41.38M	18.6%	78.61M	12755	624K
Ring, Figure 1 middle	158.0 s	83.3 s	154.9	48.89M	21.8%	61.40M	17877K	1095K
Ring, Figure 16 middle	50.7 s	19.4 s	200.4	10.70M	21.1%	10.35M	4675K	287K
Roses, Figure 16 top	68.2 s	19.0 s	158.4	5.61M	24.6%	1.64M	3405K	325K
Sphere, Figure 15 right	163.2 s	124.3 s	192.3	49.41M	12.8%	41.39M	17862K	2484K
Teapot, Figure 10	202.4 s	101.6 s	154.7	54.27M	32.3%	414.82M	22296K	1386K
Tile, Figure 14 middle	68.2 s	19.0 s	199.6	5.71M	13.5%	6.68M	2156K	295K
Tile, Figure 14 bottom	23.7 s	13.7 s	199.7	5.27M	11.8%	11.14M	2331K	260K
Vase, Figure 7	198.1 s	158.7 s	184.4	67.72M	18.4%	94.80M	22616K	2878K
Vase, Figure 11	208.6 s	135.4 s	139.5	68.29M	16.7%	418.13M	25564K	1985K
Vase, Figure 14 top	68.6 s	41.8 s	184.4	25.87M	17.4%	7.08M	7960K	788K
Vase, Figure 17 right	200.6 s	98.2 s	162.0	72.94M	16.9%	79.91M	23445K	1249K
Vase, Figure 17 left	287.5 s	204.2 s	170.1	64.90M	17.1%	61.49M	23437K	3755K

For each model, we list the average diagonal of the voxelizations representing the decorations, assuming the voxel's edges unitary, the number of voxels of the base shape and the decorations (in millions), the number of overlapping voxels in the decorations, for which our system solves and deform, the number of triangles of the marching cubes' extracted mesh (in thousands), and the size of the final mesh of the decorations alone (in thousand triangles, excluding the base).

Finally, when using physics simulators, we also found the need to perform significant parameter tuning. Indeed, the two examples presented above required different constraint values set through dozens of hours of trial-and-error. Moreover the behavior is geometry-dependent, and using a different level of detail changes the simulation result. As a consequence, the trial-and-error to set the parameters must be done on the final resolution, and each attempt takes hours. This process is tedious, but must be done thoroughly to avoid poor configuration that can lead to wrong material behavior, such as the one shown in Figure 22 in which the decorations lose their shape after the inflation phase. Furthermore, while the domain of possible parameters is wide, many configurations are not stable. In Figure 22, we show a simulation stopped half way through, after a failure of the simulator. Interestingly enough, using the same parameters and mesh resolution, but changing the number of decorations, can result in a successful simulation. However, PAVEL does not require parameter tuning and while it is possible to alter the material behavior changing Equation (1), all the results presented in the article use the same parameters.

In conclusion, we found that the physics simulators tested are not easily adapted to produce results similar to our own, are significantly slower, and require extensive parameter tuning.

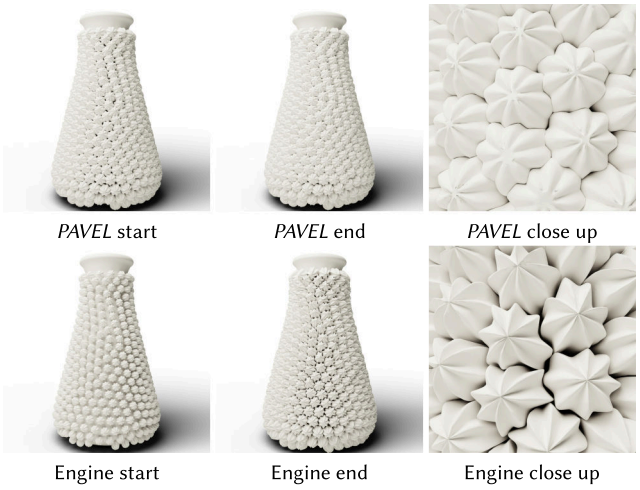


Fig. 20. Comparison of results obtained with PAVEL and a XPBD physics simulator [Macklin et al. 2016] for the same decorations placement used in Figure 17.

9 DISCUSSION

To the best of our knowledge, *PAVEL* is the first attempt to propose a method to create packed volumetric decorations on 3D models. This allows us to create complex handmade-like surface patterns without time-consuming procedures such as sculpting or simulating in detail the artistic procedure, keeping user control on the properties of the distribution of the elements on the surface.

Throughout the article, we demonstrate that we can visually simulate real-world hand-crafted objects. We do so by placing decorative elements with point-sampled uniform or stripe-align distributions and packing elements with a volume-preserving deformation. The decorated models can be used for virtual applications of 3D printed for tangible reproductions.

Table 3. Performance Comparison between *PAVEl* and an XPDB Physics Simulator [Macklin et al. 2016]

Model	*PAVEL* time	Simulator time	Performance improvement	*PAVEL* voxels	Simulator triangles
Vase, Figure 20	8 mins	356 mins	45×	64.9M	8.9M
Tile, Figure 21	1.5 mins	134 mins	89×	5.71M	847K

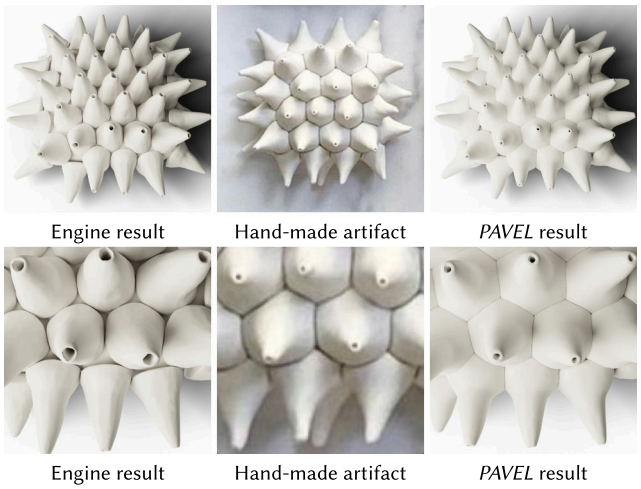


Fig. 21. Comparison of our results and the one obtained with a XPBD physics simulator [Macklin et al. 2016] when reproducing a real-world hand-made artifact.

Limitations and future work. Our method has two main limitations. First, the automatic placement of decorations is limited to approximately isotropic elements or elongated shapes placed on stripes. As stated in Section 1, this is a design choice to simplify the packing procedure, which is in general NP-hard, leaving to the deformation step to fill the gaps between elements.

However, we plan in future work to improve the initial packing step with novel options, as many other initialization choices can be adopted. Elements with constant anisotropy could be, for example, placed using the CVT variant described in Lévy and Liu [2010]. The placement could also modified optimizing elements' position and orientation after the initial placement to enhance the quality of the packing of non-radially symmetric elements. Elements with specific shapes could be also placed based on examples using optimization procedures forcing neighborhood similarity, as done in Ma et al. [2011b].

The addition of many different initialization methods would make the system complex, while our goal is to create a design tool allowing an easy control with a few parameters. For this reason, we plan to investigate on a possible semantic description of element-based textures [Godi et al. 2019] to find a few control parameters related to elements and spatial distributions.

The second limitation is our use of fast marching, which limits the complexity of the deformed objects' details, since the number of voxels quickly grows with respect to surface details. In the future, we plan to explore sparse representations based on spatial hashing and wide-branching trees that are of common use in 3D animation.

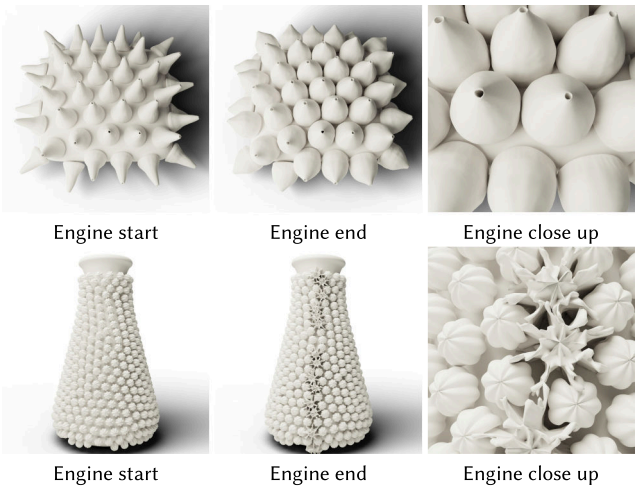


Fig. 22. Physics simulators [Macklin et al. 2016] require significant parameter tuning for our configurations to avoid artifacts. Top: A proper ratio between constraints strength is needed to maintain decorations' shapes while inflating. Bottom: Artifacts due to the interaction between collision handling and shape preservation.

In addition to the improvements listed above, we plan to improve the pipeline by adding more options to control the design, namely, the possibility of performing automatic seeding for specific decorations on different parts of the mesh.

Another planned work is the optimization of the shell decomposition for printing. We plan to develop methods for the automatic decomposition of the *PAVEL* decoration in printer-friendly patches of controlled size. We will also investigate the possibility of adding printability constraints in the fast marching evolution.

Last, we also plan to work on sparse element representation, e.g., OpenVDB. This would allow us to dramatically increase the resolution of the decorative elements, while keeping the memory footprint, and potentially the execution times, low.

ACKNOWLEDGMENTS

The authors gratefully recognize the inspiration they had looking at the work of Heather Knight [Knight 2021] and wish to thank her for the valuable feedback on the results.

REFERENCES

Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi. 2017. Sketch-based implicit blending. *ACM Trans. Graph.* 36, 6 (2017).

Philip T. Barton, B. Obadia, and Dimitris Drikakis. 2011. A conservative level-set based method for compressible solid/fluid problems on fixed grids. *J. Comput. Phys.* 230, 21 (2011), 7867–7890.

Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. 2014. A survey on position-based simulation methods in computer graphics. *Comput. Graph. Forum* 33, 6 (2014), 228–251.

Pravin Bhat, Stephen Ingram, and Greg Turk. 2004. Geometric texture synthesis by example. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP'04)*. Association for Computing Machinery, New York, NY, 41–44.

Foundation Blender. 2020. Blender. Retrieved from https://www.blender.org.

Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *Proceedings of the ACM SIGGRAPH 2007 Sketches* Conference.

Weikai Chen, Yuexin Ma, Sylvain Lefebvre, Shiqing Xin, Jonàs Martínez, and Wenping Wang. 2017. Fabricable tile decors. *ACM Trans. Graph.* 36, 6 (2017).

Weikai Chen, Xiaolong Xia, Shiqing Xin, Yang Xia, Sylvain Lefebvre, and Wenping Wang. 2016. Synthesis of filigrees for digital fabrication. *ACM Trans. Graph.* 35, 4 (2016).

David Coeurjolly, Pierre Gueth, and Jacques-Olivier Lachaud. 2018. Regularization of voxel art. In *Proceedings of the ACM SIGGRAPH Talk Conference*. 44:1–44:2.

Mathieu Desbrun and Marie-Paule Cani. 1998. Active implicit surface for animation. In *Proceedings of the Graphics Interface Conference*. 143–150.

Jens Egeblad, Benny K. Nielsen, and Marcus Brazil. 2009. Translational packing of arbitrary polytopes. *Comput. Geom.* 42, 4 (2009), 269–288.

Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M. Kaufman. 2021. Guaranteed globally injective 3D deformation processing. *ACM Trans. Graph.* 40, 4, Article 75 (August 2021), 13 pages. DOI: https://doi.org/10.1145/3450626.3459757

Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. 1995. Cellular texture generation. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'95)*. Association for Computing Machinery, New York, NY, 239–248.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: A practical Gauss-Seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (2016).

Marie-Paule Gascuel. 1993. An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of the of ACM SIGGRAPH Conference*. 313–320.

Marco Godi, Christian Joppi, Andrea Giachetti, Fabio Pellacini, and Marco Cristani. 2019. Texel-att: Representing and classifying element-based textures by attributes. In *Proceedings of the 30th British Machine Vision Conference*. BMVA Press, 286.

Heather Knight. 2021. Element Clay Studio. Retrieved from https://www.elementclaystudio.com.

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe patterns on surfaces. *ACM Trans. Graph.* 34, 4 (2015).

Kin Chung Kwan, Lok Tsun Sinn, Chu Han, Tien-Tsin Wong, and Chi-Wing Fu. 2016. Pyramid of arclength descriptor for generating collage of shapes. *ACM Trans. Graph.ics* 35, 6 (2016).

Bruno Levy. 2020. Geogram. Retrieved from http://alice.loria.fr/software/geogram/doc/html/index.html.

Bruno Lévy and Yang Liu. 2010. Lp centroidal Voronoi tessellation and its applications. *ACM Trans. Graph.* 29, 4 (2010), 1–11.

Xiao Liu, Jia-Min Liu, An-Xi Cao, and Zhuang-Le Yao. 2015. HAPE3D—A new constructive algorithm for the 3D irregular packing problem. *Front. Inf. Technol. Electron. Eng.* 16, 5 (2015), 380–390.

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm *(SIGGRAPH'87)*. Association for Computing Machinery, New York, NY, 163–169.

Jérémie Dumas, An Lu, Sylvain Lefebvre, Jun Wu, and Christian Dick. 2015. By-example synthesis of structurally sound patterns. *ACM Trans. Graph.* 34, 4, Article 137 (August 2015), 12 pages. DOI: https://doi.org/10.1145/2766984

Chongyang Ma, Li-Yi Wei, and Xin Tong. 2011a. Discrete element textures. *ACM Trans. Graph.* 30, 4 (July 2011).

Chongyang Ma, Li-Yi Wei, and Xin Tong. 2011b. Discrete element textures. *ACM Trans. Graph.* 30, 4 (2011).

Y. Ma, Z. Chen, W. Hu, and W. Wang. 2018. Packing irregular objects in 3D space via hybrid optimization. *Comput. Graph. Forum* 37, 5 (2018), 49–59.

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games (MIG'16)*. 49–54.

Ravi Malladi, James A. Sethian, and Baba C. Vemuri. 1995. Shape modeling with front propagation: A level set approach. *IEEE Trans. Pattern Anal. Mach. Intell.* 17, 2 (1995), 158–175.

Stanley Osher and James A Sethian. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.* 79, 1 (1988).

Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. 2005. Shell Maps. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH'05)*. Association for Computing Machinery, New York, NY, 626–633.

Bernhard Reinert, Tobias Ritschel, and Hans-Peter Seidel. 2013. Interactive by-example design of artistic packing layouts. *ACM Trans. Graph.* 31, 6 (2013).

T. Romanova, J. Bennell, Y. Stoyan, and A. Pankratov. 2018. Packing of concave polyhedra with continuous rotations using nonlinear optimisation. *Eur. J. Operat. Res.* 268, 1 (2018), 37–53.

Christian Santoni, Claudio Calabrese, Francesco Di Renzo, and Fabio Pellacini. 2016. SculptStat: Statistical Analysis of Digital Sculpting Workflows. (2016). arXiv:cs.GR/1601.07765.

Reza A. Saputra, Craig S. Kaplan, and Paul Asente. 2021. Improved deformation-driven element packing with repulsionPak. *IEEE Transactions on Visualization and Computer Graphics* 27, 4 (2021), 2396–2408. DOI: 10.1109/TVCG.2019.2950235

Team Scikit FMM. 2020. Scikit FMM. Retrieved from https://github.com/scikit-fmm/scikit-fmm.

James A. Sethian. 1996. A fast marching level set method for monotonically advancing fronts. *Proc.. Nat. Acad. Sci.* 93, 4 (1996), 1591–1595.

Software SideFX. 2020. Houdini. Retrieved from https://www.sidefx.com.

Michael Dawson-Haggerty et al. 2020. Trimesh. Retrieved from https://trimsh.org/.

Shi-Qing Xin, Bruno Lévy, Zhonggui Chen, Lei Chu, Yaohui Yu, Changhe Tu, and Wenping Wang. 2016. Centroidal power diagrams with capacity constraints: Computation, applications, and extension. *ACM Trans. Graph.* 35, 6 (2016).

Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. 2009. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. *Comput. Graph. Forum* 28, 5 (2009), 1445–1454.

Jianming Yang and Frederick Stern. 2017. A highly scalable massively parallel fast marching method for the Eikonal equation. *J. Comput. Phys.* 332 (2017), 333–362.

Yu-Wei Zhang, Jing Wu, Zhongping Ji, Mingqiang Wei, and Caiming Zhang. 2019. Computer-assisted Relief Modelling: A Comprehensive Survey. *Computer Graphics Forum* 38 (2019), 521–534. https://doi.org/10.1111/cgf.13655

Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. 2006. Mesh quilting for geometric texture synthesis. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH'06)*. Association for Computing Machinery, New York, NY, 690–697.

Received April 2021; revised November 2021; accepted November 2021