Deformation-Driven Element Packings

Reza Adhitya Saputra

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

June 25, 2018

1 Introduction

A *packing* is an arrangement of 2D geometric *elements* within a *container* region in the plane. Fig. 1 shows an example of a packing drawn by an artist. Packings are popular in art, ornamentation, and graphics design. They can effectively convey a relationship between a unified whole (the container shape) and its many sub-parts (the elements).

Elements are shapes like animals, plants, geometric forms, or man-made objects. An artist distributes elements so that they communicate the shape of the container. The subset of the container that does not belong to any element is called *negative space* (Fig. 1, right). We can also interpret negative space as separation and gaps between elements. The evenness of negative space plays an important role in packings. The artist should arrange elements in a way that their boundaries interlock with each other, causing the separation between neighbouring elements to become roughly the same everywhere. As the elements in a packing become perfectly interlocked, the packing turns into a *tiling*: a set of elements that exactly fill a container with no overlaps and no negative space (Fig. 2b).

There has been a moderate amount of past research in computer graphics, particularly in the field of non-photorealistic rendering, on the generation of packings, tilings, or mosaics. See Section 2 for specific examples. However, most techniques pack elements via rigid transformations, leading to uneven element distribution and overlaps. Jigsaw Image Mosaics [KP02] and collages based on the Pyramid of Arclength Descriptor [KSH+16], might be described as *data-driven*. These techniques rely on assembling a large library of elements, so that given an area to fill in a partial composition, there is likely to be an element in the library with a compatible shape. The challenge is *finding compatible*

Figure 1: A packing created by hand by an artist, depicting autumn-themed elements (left), together with a visualization of its negative space (right). (Artist: balabolka on Shutterstock)

elements, which requires designing a shape descriptor that is fast and robust. These techniques suppress imperfections by deforming elements in a final post-processing step after they freeze the element positions. The data-driven approaches do not always work. They cannot guarantee to find a compatible element at every iteration, and elements typically do not fit perfectly with each other or the container boundary. The remedy here is providing more data with increased computation time. However, a large library may not be feasible or artistically desirable. If an artist wants a packing of cats, and a data-driven approach cannot find a good result with ten cat shapes, the artist may not want to draw 100 or 1000 cat shapes to ensure a better fit.

We propose *deformation-driven* methods. Instead of finding compatible elements, we intend to *create compatible elements* through element deformation that can adapt to the shapes of neighboring elements and the container boundary. We allow elements to deform in a controlled way, to trade off between the evenness of the element distribution and the deformations of the individual elements. By building an algorithm with a controllable deformation model at its core, we achieve a more even distribution of negative space, even with a small library of element shapes.

Deformation-driven methods also allow us to work toward a design principle called *uniformity amidst variety* [Hut29, Gom84]. *Uniformity* aims for an overall unity of design; *variety* seeks to break up the monotony of pure repetition. We can achieve a degree of uniformity by using repeated copies of a small library of elements, but balance that uniformity with variety by deforming those elements. We believe that there is a value in deformation that can generate plausible families of related elements from a single input shape.

The rest of this proposal is organized as follows. Section 2 presents related work. In Section 3, we discuss FLOWPAK, which deforms elements to flow along a user-defined vector field inside a container. Section 4 presents RepulsionPak, which creates packings with even negative space by utilizing repulsion forces and controllable deformation. Both FLOWPAK and RepulsionPak were previously published in computer graphics conferences, but in each case we discuss limitations and opportunities for future work. Section 5 discusses a quantitative model for evaluating packings, based on measuring the evenness of the negative space. We use a method called *spherical contact probability functions* to demonstrate the benefit of deformation-driven packings. Section 6 proposes AnimationPak, which can generate packings of elements with scripted animations. Finally, we discuss conclusions in Section 7.

2 Related work

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

Figure 2: (a) Decorative mosaic [Hau01]. (b) A tiling inspired by M.C. Escher [KS00]. (c) Jigsaw image mosaic [KP02] that is an example of a dense packing. (d) Pyramid of arclength descriptor [KSH+16] that can generate collages. (Figures ©ACM).

Rigid packing: The ever-popular Lloyd's method is an iterative process for creating a perceptually even distribution of points, and has been used in various forms in procedural packing methods. Hausner [Hau01] used a variant of Lloyd's method to pack oriented rectangles into a container region, simulating the appearance of traditional mosaics (Fig. 2a). Hiller et al. [HHD03] extended Lloyd's method to distribute polygonal elements instead of points, reducing the overlaps in Hausner's approach. Abdrashitov et al. [AGYS14] developed an interactive tool to create mosaics by threading elements along sketched paths.

Dense packing and tessellations: Gal et al. [GSP+07] used local shape descriptors on 3D models to fill a 3D container with a "collage" in the style of Arcimboldo. Huang et al. [HZZ11] produced Arcimboldo-like collages in 2D by layering objects cut out from

(a) (b)

Figure 3: (a) An animation generated by Animosaics [SLK05]. (b) An animated composition generated by FFT-based packing [DKLS06]. We show two frames for each. (Figures ©ACM)

images. These methods benefit from overlaps, which join elements into a single large object. Kaplan and Salesin [KS00, KS04] deformed one or two user-supplied input shapes into new shapes that tile the plane (Fig. 2b).

Jigsaw Image Mosaics (JIMs) [KP02] packed nearly-convex elements tightly by placing one element at a time and backtracking as needed (Fig. 2c). Kwan et al. proposed a Pyramid of Arclength Descriptor (PAD) [KSH+16] in order to find new elements that partially matched existing element boundaries as a container was being filled (Fig. 2d). Both methods permitted some elements to overlap. They could both lessen gaps and overlaps using deformation that was applied locally near edges in a post-processing step after elements were frozen in place.

Animated packings: Animosaics [SLK05] and a follow-up spectral approach [DKLS06] demonstrated the creation of animated packings using only rigid transformations. Animosaics generated animated packings by rearranging small non-deforming elements from frame to frame inside an animated container (Fig. 3a), including the possibility of elements being added or removed. The spectral approach was able to pack elements with scripted animations inside a static container. These animations become 3D extruded shapes in a spacetime coordinate system (Fig. 3b). Unlike Animosaics, the FFT-based method does not consider adding or removing elements, causing pockets of empty space that appear and disappear over time.

Discrete texture synthesis: Some past work has sought to adapt example-based texture synthesis methods from raster images to vector graphics, producing rigid distributions of elements that mimic the statistics of an exemplar. Barla et al. [BBT+06] and Ijiri et al. [IMIM08] used a growth model that copies small neighbourhoods from the exemplar into a larger output texture. Hurtut et al. [HLT+09] developed a statistical sampling

(a) (b) (c)

Figure 4: (a) Floral ornament [WZS98]. (b) DecoBrush [LBW+14]. (c) Inscribed curves on Voronoi cells [WKSM12]. (Figures ©ACM)

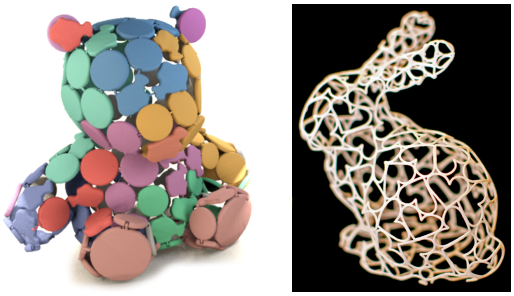
method based on multitype point processes. AlMeraj et al. [AKA13] stamped out copies of the exemplar and discarded overlapping elements. These techniques are all concerned with replicating the uneven element distribution of an exemplar.

Procedural methods can be utilized to generate textures as well. Loi et al. [LHVT17] developed a method comprising of three parts: *partition operators* that determine the global arrangement of a texture, *mappers* that refine element placements locally, and *merging operators* that combine multiple patterns into one. This approach is suitable for artists who want complete control over an element distribution. Similar to other texture synthesis methods, it does not explicitly address the distribution of negative space.

Decorative and aesthetic patterns: A distinct category of past research seeks to develop explicit procedural models for authoring decorative patterns or aesthetic arrangements. Wong et al. [WZS98] articulated a set of design principles for decorative art: repetition, balance, and conformation to geometric constraints (Fig. 4a). They went on to describe a grammar-like system for laying out floral ornament. Lu et al. developed DecoBrush [LBW+14], in which ornamental elements are deformed along line art but are not required to fill containers (Fig. 4b). Wyvill et al. [WKSM12] constructed arrangements of sponge-like volumes by inscribing closed curves inside 2D Voronoi cells (Fig. 4c).

Text and letter packings: Our proposed research is also related to work on packing individual letterforms or blocks of text into container regions. Xu and Kaplan [XK07] and Zou et al. [ZCR+16] constructed *calligrams* by filling a container with a small number of deformed letters composing one or two words (Fig. 5). Their goal was to balance between filling the container and preserving readability. Because the order of the letterforms was defined by the text, their solutions usually required significant distortion of the individual letters. Maharik et al. [MBS+11] explored Digital Micrography, in which whole lines of text

Figure 5: Calligraphic packing of an *elephant* [XK07]. (Figure ©ACM).



(a) (b)

Figure 6: (a) Modular surfaces [CML+17]. (b) Ornamental curves that cover a surface [ZCT16]. (Figures ©ACM)

deform to fit along dense streamlines in a flow field. Their results more closely resemble textures than packings.

Packings for fabrication: Related work in fabrication has sought to cover surfaces with arrangements of elements. Chen et al. [CZX+16] and Bian et al. [BWL18] developed methods to synthesize intricate patterns out of a few elements. In later work, Chen et al. [CML+17] generated modular surfaces by computing contact point networks of rigid elements (Fig. 6a). Zehnder et al. [ZCT16] proposed semi-automated tools for deforming ornamental curves to cover a surface (Fig. 6b). In these cases, the layout of elements must be computed to satisfy both aesthetic and structural goals—most obviously, elements must overlap to produce a connected result that will hold together when 3D printed.

3 FLOWPAK: Flow-based element packings

Our first project was FLOWPAK [SKAM17], a technique for filling a container region with elements that are deformed to communicate a sense of directionality or flow. Elements can be oriented in the local direction of flow, but can also be deformed to capture changes in flow direction. Flow adds visual interest to a composition, engaging the viewer by providing a sense of progression and movement through elements. A hand-drawn dog example in Fig. 7 shows many elements that appear to flow outward from the flower in the centre of the torso, and then up the neck and down into the legs. Although there has been a moderate amount of past research on the generation of packings or mosaics, this past work is not appropriate for creating flow-like designs similar to the dog example.

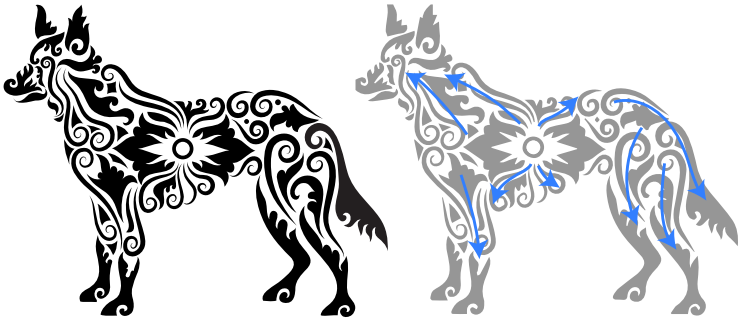


Figure 7: An example of flow visual style, including a visualization of the flow directions of elements (by ComicVector703 on Shutterstock).

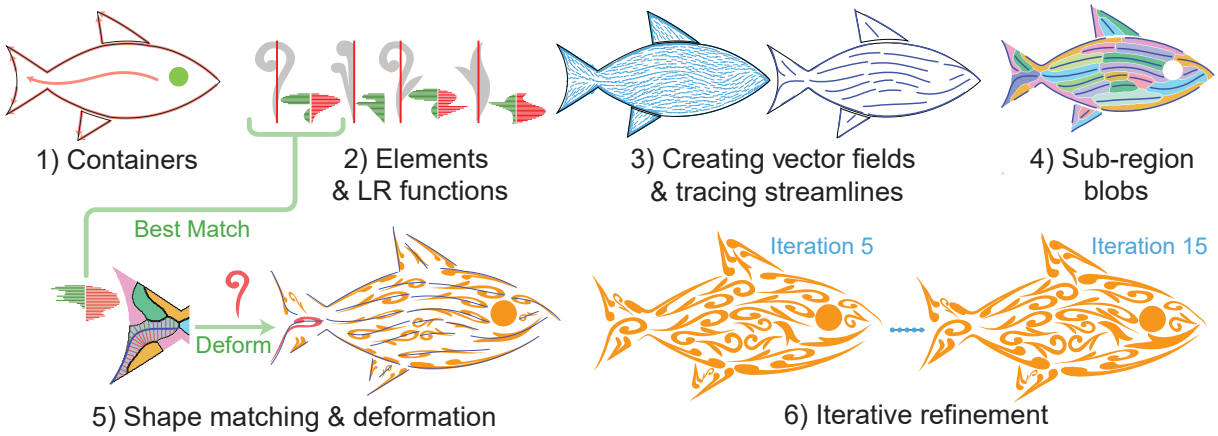
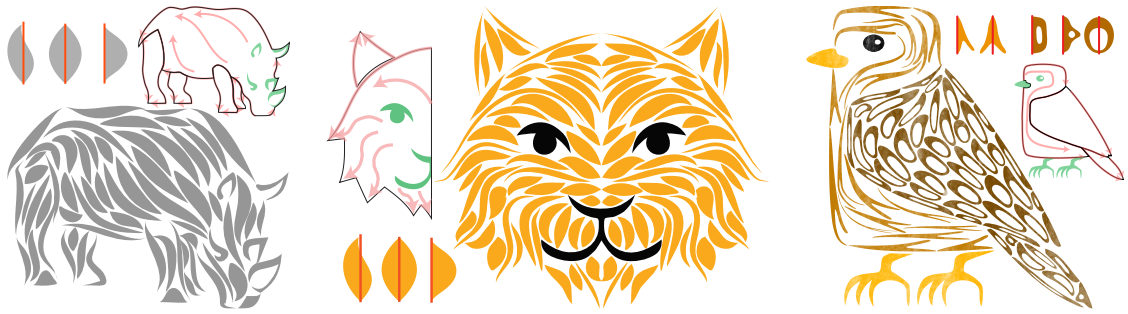


Figure 8: A visualization of the steps in FLOWPAK. The containers are shown as three black outlines in (a): the body and two fins. They are annotated with directional guides in red and fixed elements (in this case, the eye) in green.

Pipeline: We express the user's desired flow by placing evenly spaced streamlines inside the container region. Each streamline is then replaced by an element chosen from a pre-drawn set. The element is bent along the streamline to communicate flow, and also deformed to balance the usage of negative space with elements placed on adjacent streamlines. Additionally, we need to use a small number of *fixed elements* to solve specific design problems or provide focal points; for example, the dog's eye is expressed via a carefully placed spiral and specialized shapes convey the dog's paws.

Fig. 8 gives an overall view of our system. The numbers in the following steps correspond to parts of the figure.

1. Read input containers and also copy any fixed elements to the output art. *Direction*



(a) Rhinoceros (b) Cat (c) Box bird

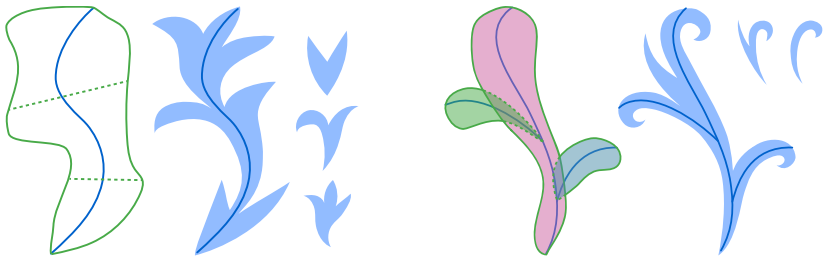
Figure 9: Three packings created using FLOWPAK

guides are shown as pink curves.

2. Create a shape descriptor for each element. An element has a *spine* (shown as a red vertical line). We call the descriptor an *LR function*; it is built by sampling the element's spine at n locations and at each location determining how far the ornament extends to the left and right of the spine.
3. Use the direction guides to create vector fields, and trace streamlines.
4. Divide the containers into *subregion blobs* around the streamlines by constructing an approximate generalized Voronoi diagram. We then compute an LR function for each blob.
5. Use LR functions to determine the best element for each blob. Place the element in the blob, treating it as a skeletal stroke [HLW93] and mapping the element's spine to the streamline.
6. Iteratively refine the element placements to eliminate empty areas by shifting streamlines, expanding blobs, and reflecting the elements.

Results: In Fig. 9a, we show a packing of a rhinoceros with simple teardrop elements that demonstrates the variety we achieve in shape and curvature. Apart from fixed elements like the horns, every packed element is a deformed copy of one of the three element shapes shown above the packing. The cat in Fig. 9b demonstrates a symmetric packing with a fur contour. We only compute the left part and reflect the result. We asked an artist to draw containers and a decorative elements, resulting the bird design in Fig. 9c. The artist requested that different elements and densities be used in different container regions; the result has sparse "Y" elements in the breast and head, and denser "O" elements in the wings.

Further improvements: We see many possibilities for further improvements to FLOWPAK:



(a) (b)

Figure 10: (a) Thread multiple elements on a single streamline. (b) Combine multiple streamlines to create branching.

1. **High curvature streamlines:** Our results do not have significant high-curvature streamlines like u-turns, since they could unpleasantly fold the decorative elements. This could be solved with a folding avoidance algorithm [Ase10] or a content-aware warping method.
2. **Automatic creation of fixed elements:** We would like to explore the automatic creation and placement of the fixed elements, perhaps by discovering them as salient regions in source photographs, and extracting and vectorizing them. This extraction must be carried out carefully, yielding enough fixed elements to communicate a container clearly without disrupting the uniformity of the design.
3. **Element threading:** In FLOWPAK, an element must completely use up its streamline, causing a few elements to stretch too long. We would like to combine multiple shorter elements by threading them along streamlines, creating a longer *compound element* (Fig. 10a). First, we need a way to connect the overlapping segments of the elements together to create smooth and plausible transitions. This possibly leads us to a deeper research topic: how to develop a content-aware blending method on vector graphics patterns. Second, we should carefully decide on the selection of elements we have to thread and the number of elements we need. We would like to investigate a greedy approach with backtracking [KP02].
4. **Branching structures:** We also would like to support branching structures to resemble floral ornaments (Fig. 10b). We start with tracing a long streamline that follows the vector field, and then generate shorter streamline branches, which produce blobs that partly overlap the main blob. Similarly, we also need to connect and blend element segments at every intersection. We also should consider the semantics of the element shapes. Not every part of an element can support branching; for example, a leaf cannot be connected with twigs. In another example, a stem or a trunk is an ideal segment where we can glue other elements.

4 RepulsionPak: Deformation-driven element packing with repulsion forces

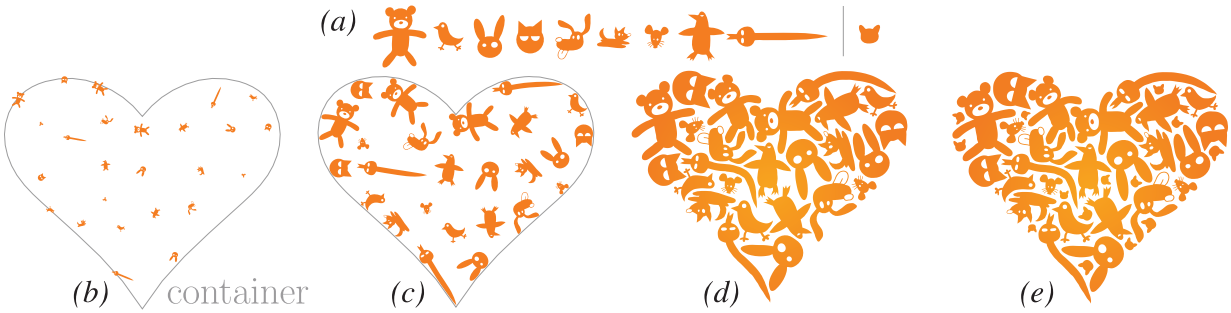


Figure 11: The pipeline of RepulsionPak. (a) A library of elements, comprising nine primary elements and a single secondary element. (b) A target container with the initial distribution of scaled-down elements. (c) The simulation in progress, showing the elements growing, translating, rotating, and deforming. (d) The resulting packing of primary elements. (e) The final result, after adding secondary elements. Fig. 12 shows the deformations of some of the elements.

Our second project was RepulsionPak [SKA18]. We construct a packing using a simple physical simulation, in which each element is represented by a mass-spring system called an *element mesh*. *Repulsion forces* between neighbouring meshes work to even out the negative space, inducing displacements in mesh springs. These displacements translate, rotate, and deform the elements as they gradually adapt to the shapes of their neighbours and the container boundary. To control the amount of deformation, we use *spring forces* within a mesh to preserve element shapes.

In Fig. 1, an artist initially distributes *primary elements* so that they communicate the shape of the container. When the primary elements leave behind large pockets of negative space, the artist typically fills those pockets with small *secondary elements*, often designed as simple shapes. The challenge—and visual appeal—of packings follows in part from aligning neighbouring elements along compatible segments of their boundaries, suggesting that they interlock by design.

Pipeline: The system requires a library of primary and optional secondary elements (Fig. 11a). RepulsionPak starts by preprocessing the elements, adding additional space around each to implement the spacing distance, and fitting a triangle mesh over each element (Fig. 13). Small copies of these *primary elements* are randomly placed in the containers (Fig. 11b). The system then performs a physics simulation on the meshes, making them simultaneously grow and repel each other. Let $\boldsymbol{x}$ be a vertex of an element mesh.

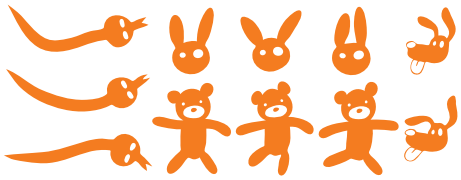


Figure 12: Some of the elements in the final packing in Fig. 11, showing the effect of deformation in our simulation.

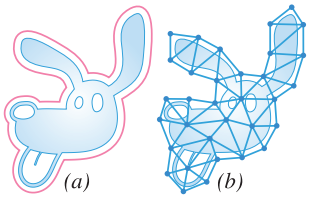


Figure 13: (a) An element with its boundary offset to create a skin, drawn in red. (b) A triangle mesh with boundary vertices on the skin.

The vertex $\boldsymbol{x}$ will experience repulsion forces from neigbouring meshes. The magnitude of a repulsion force is proportional to the inverse square of the closest distance from $\boldsymbol{x}$ to a neighbour. Therefore, the total repulsion force applied to $\boldsymbol{x}$ depends on two things: (1) the number of neighbors and (2) the proximity of these neighbors. Every vertex experiences a different force, and these differences cause the whole mesh to deform.

After each iteration of the simulation, meshes grow into adjacent space if possible, so that they gradually consume the negative space in the container. We stop the simulation when the element meshes are no longer able to maneuver enough to consume the remaining negative space (Fig. 11d). An optional second simulation further reduces and evens out the negative space. It begins by placing small *secondary elements* in large pockets of negative space. This simulation is the same as the first, except that vertices of primary element meshes are not allowed to move (Fig. 11e).

Results: The animal packing in Fig. 11 has several elements with limbs (the bear, fox, chick, and penguin), extensions (the dog and bunny ears), and long shapes (the snake). Fig. 12 highlights the deformations for some of these elements; they are noticeably more deformed than nearly-convex elements like the cats and mice. The packing on the left in Fig. 14 exhibits significant deformation in the wings and the tails of the birds. The middle packing in Fig. 14 shows significant deformation in the wings of the bats. The butterfly packing on the right in Fig. 14 is an attempt to reproduce the visual style of a dense packing. The target container is made from two regions, one with internal holes. The resulting packing is tight but overlap-free.

Proposed ideas: We see many possibilities for further improvements to RepulsionPak:

1. **Non-random initial configuration:** It would be interesting to generate an initial element placement for RepulsionPak based on shape matching, instead of assigning elements to sample positions at random. RepulsionPak may have difficulty in handling sharp narrow features of the container boundary. For example, a round-shaped

Figure 14: Three packings created using RepulsionPak: Birds, Bats, and Butterflies.

element cannot fill a pointy corner. A better solution will depend on the element library but also on the initial placement of elements. We can benefit from a shape matching algorithm by placing a few elements to cover key boundary features then running the rest of the initial placement at random.

2. **Interactive packings:** We would like to develop human-in-the-loop interactions, in which the user and the computer work together to create a desired composition. Examples of recent work in this style include research by Zehnder et al. [ZCT16] and Gieseke et al. [GALF17], which let the user directly manipulate elements while a composition is being created. It would also be interesting to display an interactive packing on a large touch screen, and let people to move elements around by touching and dragging gestures.
3. **Fabrication:** We would like to explore the use of RepulsionPak in a fabrication context. For example, our boundary compatibilities might be used to locate good contact points to create a connected object. Alternatively, it would be interesting to 3D print the negative space, which is already connected, leaving holes that surround the element shapes. Another potential application is to use our algorithm in the context of CNC machining to reduce the amount of waste material.
4. **Better physics-based methods:** To simulate deformable elements, RepulsionPak uses fast and simple mass-spring systems with explicit Euler. However, contemporary research has yielded many more sophisticated physical simulation methods, such as Finite Element Methods, Position Based Dynamics [MHHR07], and Projective Dynamics [BML+14]. Although our main goal was to demonstrate the validity

of a deformation-driven approach, and not to contribute a new physical simulation method, we intend to experiment with several to investigate if any offers a suitable trade-off between performance and quality.

5 Measuring the evenness of negative space

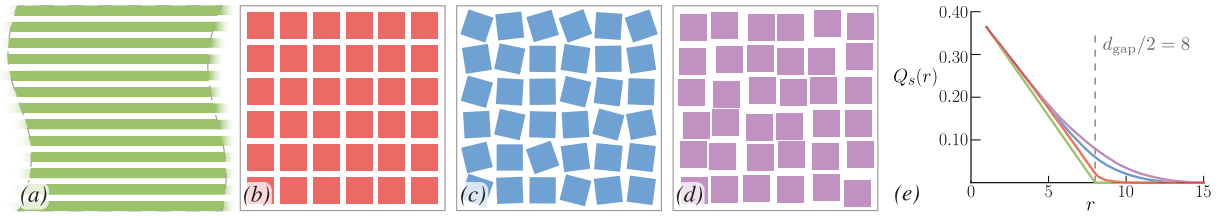


Figure 15: Spherical contact probabilities for reference packings. A "perfect packing" of infinite stripes is shown in (a), followed by a square packing with the same area fraction and d_{gap} in (b). The square packing is then perturbed with random rotations in (c) and translations in (d). The corresponding $Q_s(r)$ functions are plotted in (e).

As with many techniques in artistic applications of computer graphics, evaluating the quality of a computer-generated element packing is a challenge. We would like to investigate the use of quantitative measures of the evenness of a packing's negative space, which we hypothesize plays a major role in the perceived quality of a packing.

We use a method called the *spherical contact probability* (SCP), denoted as $Q_s(r)$, which is the probability that a disc of radius r, chosen uniformly at random within the container region, lies entirely within the packing's negative space. In spatial statistics, the SCP is closely related to *spherical contact distribution function* [CSKM13]: $H_s(r) = 1 - \frac{Q_s(r)}{Q_s(0)}$. Note that $Q_s(0)$ is the probability that a random point lies in negative space, and must therefore be equal to $1 - A$, where A is the packing's area fraction.

In order to interpret the SCP correctly, it is helpful first to examine a "packing" with perfectly even negative space (Fig. 15a). Consider a pattern of infinite horizontal stripes of width d_{s}, separated from each other by distance d_{gap}. The area fraction of the negative space is $Q_s(0) = d_{\text{gap}}/(d_{\text{gap}} + d_{\text{s}})$; It is clear that $Q_s(d_{\text{gap}}/2) = 0$, because no disc of diameter greater than d_{gap} can fit in the negative space (our d_{gap} is twice the radius of the disc). Furthermore, $Q_s(r)$ will decrease linearly between these two values, and remain at zero thereafter; its graph will consist of a tilted line segment connected to a horizontal ray.

No real-world packing exhibits this SCP. Even in a perfect arrangement of squares (Fig. 15b), the intersections of horizontal and vertical channels produce pockets of negative

Figure 16: A packing created by RepulsionPak using the elements from Fig. 1. Our SCP is lower than the artist's result and uses fewer secondary elements.

space that can accommodate discs of diameter $\sqrt{2}d_{\text{gap}}$. These pockets tend to raise the SCP slightly everywhere, and cause it to bend into a small tail that approaches zero gradually. In a more general packing, we may think of d_{gap} as representing the desired width of the negative space. For a given set of elements in a container, the best packings should have an SCP that stays close to the idealized stripe function most of the way down, attain a low value at $r = d_{\text{gap}}/2$, and then bends towards horizontal near that value. In less effective packings (Fig. 15c,d), the negative space will be narrower in some places and wider in others, recognizable as a shallower SCP with a longer tail.

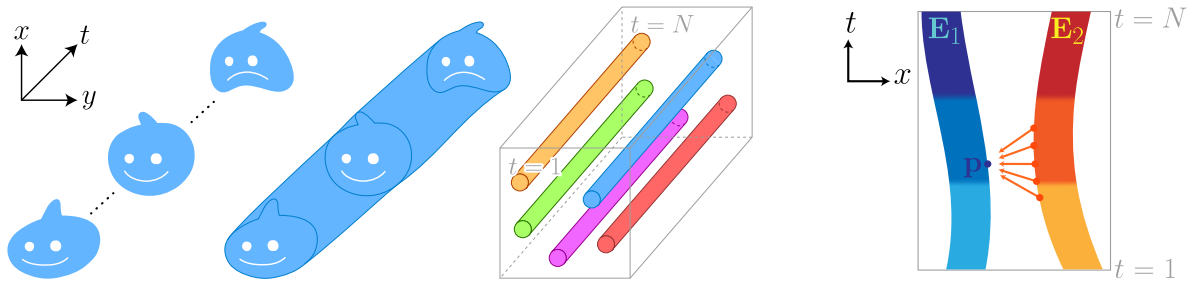
Comparison to an artist-made packing: Fig. 16 shows RepulsionPak's results packing the elements from the artist-made Fig. 1. We estimated the average separation d_{gap} between elements in the artist-made packing, and used the same d_{gap} to generate ours. The graph compares the SCP of our result with that of the artist's, showing that our negative space is less. Our result also has fewer large empty gaps, as indicated by the inclusion of fewer secondary elements.

Criticisms of SCP: The idea of using SCPs to evaluate the evenness of negative space is promising, but it is more delicate than we understood. Our result in Fig. 16 is more crowded than the artist's, suggesting that the way we estimate d_{gap} requires improvement. The challenge is to compute d_{gap} for a given packing with no additional information about how that packing was produced. It is possible to calculate ridges between elements using a generalized Voronoi diagram. The intended d_{gap} should appear as a mode in that distribution. Alternatively, we might not need to estimate the artist's d_{gap} since we only need to create a packing that has the same total area of negative space, at which point it becomes fair to use SCPs.

We focus on the evenness of negative space because it is something that we can measure. Ultimately, we want to understand packing quality directly, and SCPs may simply be a proxy for that. A broader research project would be to study experimentally how well SCPs actually correlate with the perceived quality of a packing. It would also be great to

delve more deeply into goals the artist has and ask what they are trying to achieve.

6 AnimationPak: Packing elements with scripted animations

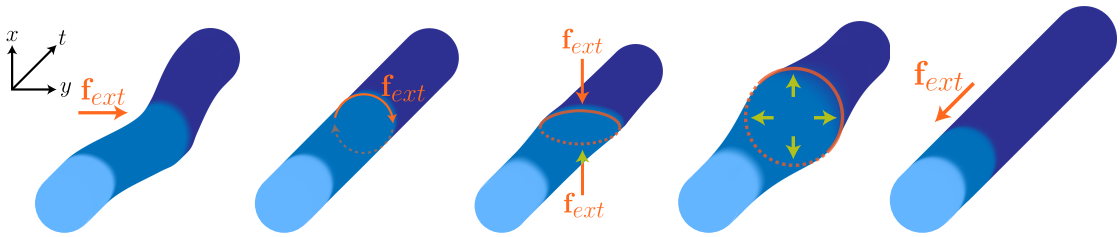


(a) (b) (c) (d)

Figure 17: AnimationPak. (a) An animator designs a character with a scripted animation. (b) The character is a 3D continuous element in a spacetime coordinate system. (c) An initial random placement that is analogous to Fig. 11b. For simplification, we draw the elements as tubes, although in practice these shapes are more complex. The container is shown as a 2D grey square that is extruded so it becomes a spacetime box from $t = 1$ to $t = N$. (d) Multiple points on element $\mathbf{E}_2$ generates repulsion forces applied to a point $\mathbf{p}$ on element $\mathbf{E}_1$.

As most packing or mosaicking techniques focused on generating static arrangements, we would like to study dynamic animated packings. For example, packing elements with animations, arranging elements inside an animated container, or both. To the best of our knowledge, the research topic of animated packings has received little attention, with previous work limited to Animosaics [SLK05] and the spectral approach [DKLS06]. Additionally, there is a great opportunity to use the deformation-driven approach to generate high quality animated packings with a distribution of negative space that remains even over time.

AnimationPak is a 3D analogue to RepulsionPak that generates packings with animated elements, for example, packings of swimming fish or flapping butterflies. we plan to develop AnimationPak to pack animated elements via controllable deformations, trading off between the evenness of the packing and preserving the shapes of elements along with their animations. We should eliminate pockets of empty space that appear and disappear over time using deformations, but in a controlled way so we do not create unpleasant distor-



(a) Bending (b) Twisting (c) Distortion (d) Growth (e) Monotonic deformation in t-axis

Figure 18: Examples of deformation.

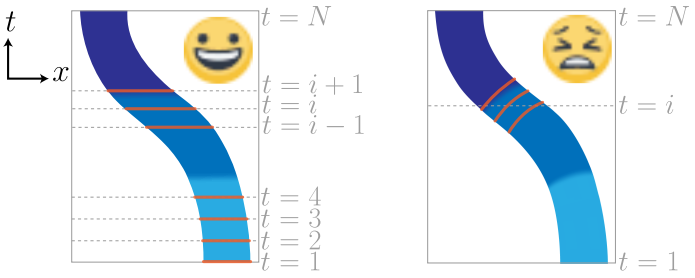
tions of the elements. The resulting animated packing can then be rendered slice-by-slice as an animated 2D packing.

AnimationPak requires input animated elements (Fig. 17a); each has a scripted animation that is generated using a standard 2D animation tool or a procedural method. This animation becomes a 3D extrusion in a spacetime coordinate system (Fig. 17b). We refer to the resulting 3D structure as a *spacetime element*. An animator may create an animation by manipulating the 2D element by deformation, scaling, or rotation. However, we assume that the animator does not use translation so that the extrusion of an element is generally straight. Let us define a mathematical description of a spacetime element as a 3D continuous elastic body. We define $\mathbf{E}(\mathbf{p})$ as an indicator function to determine whether a point $\mathbf{p} = (x, y, t) \in \mathbb{R}^3$ is inside an undeformed element. The value of $\mathbf{E}(\mathbf{p})$ is 1 if $\mathbf{p}$ is inside the element, and 0 otherwise.

Initially, small copies of elements are shrunk and randomly placed inside a container (Fig. 17c). Since the extrusions of elements are approximately straight, we can ensure that the initial placement is overlap free. We perform a physics simulation on the spacetime elements, making them simultaneously grow, deform, and repel each other. Fig. 17d shows an illustration of repulsion forces. When the element undergoes deformation, every material point $\mathbf{p}$ is displaced to a new location $\mathbf{p}'$. The relation between each $\mathbf{p}$ and $\mathbf{p}'$ is captured by a *deformation map* $\phi(\mathbf{p}) : \mathbb{R}^3 \to \mathbb{R}^3$. Each element has its own deformation map. The new deformed point can be calculated as $\mathbf{p}' = \phi(\mathbf{p})$.

We require the deformation map ϕ to satisfy the following constraints:

1. **The mapping ϕ should be a homeomorphism**: it must be continuous and one-to-one, with a continuous inverse ϕ^{-1}. In that case, we may then say that a point p' lies in the deformed element if $\mathbf{E}(\phi^{-1}(\mathbf{p}')) = 1$. Several examples of possible deformation applied to an element are bending, twisting, general distortion, and growth (Fig. 18a-d).



(a) Rendering with time constraint

(b) Rendering without time constraint

Figure 19: Discretization and rendering. (a) Slicing elements for rendering. The time constraint requires every slice to remain parallel with xy-plane. (b) A rendering artifact on the unconstrained element.

2. **The mapping ϕ is monotonic in the t-axis**: We are allowed to compress or stretch animation segments but we should preserve the order of the entire animation. Fig. 18e shows an example where light blue and blue segments are compressed (animation speed is increased), and the dark blue segment is stretched (animation speed is decreased).
3. **Time constraint**: All undeformed points that are at the same time location should always be at the same new time location after deformation. Consider any two points $\mathbf{p} = (x_p, y_p, t_p)$ and $\mathbf{q} = (x_q, y_q, t_q)$. Their deformed locations are $\phi(\mathbf{p}) = (x'_p, y'_p, t'_p)$ and $\phi(\mathbf{q}) = (x'_q, y'_q, t'_q)$. We require that if $t_p = t_q$, then $t'_p = t'_q$. This constraint allows us to render the 2D element in each frame correctly (Fig. 19a). An unconstrained element would cause a rendering artifact because materials that are originally from different time locations would get mixed together (Fig. 19b).

Animated container: We consider a more difficult case where the container itself is animated, in which cases we may want to add or remove elements during the course of the animation since the 2D area of the container changes over time. If the area gets smaller, the elements naturally get shrunk and we may want to remove a few elements. If the area is bigger, we should make the elements bigger and we may want to add new elements.

Discretization and rendering: We cut each element into *slices* $\{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, ..., \mathbf{S}_N\}$ (Fig. 19a). We define a slice $\mathbf{S}_i$ as a 2D cross-section that is created by cutting through all material points at $t = i$. We also cut the container $\mathbf{C}$ into container slices $\{\mathbf{C}_1, \mathbf{C}_2, ...\mathbf{C}_N\}$. The container is animated if each $\mathbf{C}_i$ has a different 2D area or their 2D boundaries are different over time. The number of the slices N depends on the desired frame-rate and the length of the animation video.

7 Conclusions

We summarized the research topic of element packing. FLOWPAK was able to generate packings with flow-like patterns. RepulsionPak could create packings with even negative space through repulsion forces and controllable deformation. In addition, we discussed spherical contact probability that measures the evenness of negative space. We proposed improvements for our past work and we introduced a plan for a follow-up project called AnimationPak that can pack elements with scripted animations.

References

[AGYS14] Rinat Abdrashitov, Emilie Guy, JiaXian Yao, and Karan Singh. Mosaic: Sketch-based interface for creating digital decorative mosaics. In *Proceedings of the 4th Joint Symposium on Computational Aesthetics, Non-Photorealistic Animation and Rendering, and Sketch-Based Interfaces and Modeling*, SBIM '14, pages 5–10, New York, NY, USA, 2014. ACM.

[AKA13] Zainab AlMeraj, Craig S. Kaplan, and Paul Asente. Patch-based geometric texture synthesis. In *Proceedings of the Symposium on Computational Aesthetics*, CAE '13, pages 15–19, New York, NY, USA, 2013. ACM.

[Ase10] Paul J. Asente. Folding Avoidance in Skeletal Strokes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*. The Eurographics Association, 2010.

[BBT+06] Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. of Eurographics 2006)*, volume 25, 2006.

[BML+14] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11, July 2014.

[BWL18] Xiaojun Bian, Li-Yi Wei, and Sylvain Lefebvre. Tile-based Pattern Design with Topology Control. *Graph. Interact. Tech. Proc. ACM Comput. Graph. Interact. Tech*, 1:23 – 38, 2018.

[CML+17] Weikai Chen, Yuexin Ma, Sylvain Lefebvre, Shiqing Xin, Jonàs Martínez, and wenping wang. Fabricable tile decors. *ACM Trans. Graph.*, 36(6):175:1–175:15, November 2017.

[CSKM13] S.N. Chiu, D. Stoyan, W.S. Kendall, and J. Mecke. *Stochastic Geometry and Its Applications*. Wiley Series in Probability and Statistics. Wiley, 2013.

[CZX+16] Weikai Chen, Xiaolong Zhang, Shiqing Xin, Yang Xia, Sylvain Lefebvre, and Wenping Wang. Synthesis of filigrees for digital fabrication. *ACM Trans. Graph.*, 35(4):98:1–98:13, July 2016.

[DKLS06] Ketan Dalal, Allison W. Klein, Yunjun Liu, and Kaleigh Smith. A spectral approach to NPR packing. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 71–78, New York, NY, USA, 2006. ACM.

[GALF17] Lena Gieseke, Paul Asente, Jingwan Lu, and Martin Fuchs. Organized order in ornamentation. In *Proceedings of the Symposium on Computational Aesthetics*, CAE '17, pages 4:1–4:9, New York, NY, USA, 2017. ACM.

[Gom84] Ernst Hans Gombrich. *The Sense of Order: A Study in the Psychology of Decorative Art*. Phaidon Press Limited, 1984.

[GSP+07] Ran Gal, Olga Sorkine, Tiberiu Popa, Alla Sheffer, and Daniel Cohen-Or. 3D collage: Expressive non-realistic modeling. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '07, pages 7–14, New York, NY, USA, 2007. ACM.

[Hau01] Alejo Hausner. Simulating decorative mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 573–580, New York, NY, USA, 2001. ACM.

[HHD03] Stefan Hiller, Heino Hellwig, and Oliver Deussen. Beyond Stippling - Methods for Distributing Objects on the Plane. *Computer Graphics Forum*, 2003.

[HLT+09] T. Hurtut, P.-E. Landes, J. Thollot, Y. Gousseau, R. Drouillhet, and J.-F. Coeurjolly. Appearance-guided synthesis of element arrangements by example. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '09, pages 51–60, New York, NY, USA, 2009. ACM.

[HLW93] S. C. Hsu, I. H. H. Lee, and N. E. Wiseman. Skeletal strokes. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, UIST '93, pages 197–206, New York, NY, USA, 1993. ACM.

[Hut29] F. Hutcheson. *An Inquiry Into the Original of Our Ideas of Beauty and Virtue*. J. and J. Knapton and others, 1729.

[HZZ11] Hua Huang, Lei Zhang, and Hong-Chao Zhang. Arcimboldo-like collage using internet images. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 155:1–155:8, New York, NY, USA, 2011. ACM.

[IMIM08] Takashi Ijiri, Radomír Měch, Takeo Igarashi, and Gavin S. P. Miller. An example-based procedural system for element arrangement. *Comput. Graph. Forum*, 27:429–436, 2008.

[KP02] Junhwan Kim and Fabio Pellacini. Jigsaw image mosaics. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 657–664, New York, NY, USA, 2002. ACM.

[KS00] Craig S. Kaplan and David H. Salesin. Escherization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 499–510, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[KS04] Craig S. Kaplan and David H. Salesin. Dihedral escherization. In *Proceedings of Graphics Interface 2004*, GI '04, pages 255–262, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[KSH+16] Kin Chung Kwan, Lok Tsun Sinn, Chu Han, Tien-Tsin Wong, and Chi-Wing Fu. Pyramid of arclength descriptor for generating collage of shapes. *ACM Trans. Graph.*, 35(6):229:1–229:12, November 2016.

[LBW+14] Jingwan Lu, Connelly Barnes, Connie Wan, Paul Asente, Radomír Měch, and Adam Finkelstein. DecoBrush: Drawing structured decorative patterns by example. *ACM Trans. Graph.*, 33(4):90:1–90:9, July 2014.

[LHVT17] Hugo Loi, Thomas Hurtut, Romain Vergne, and Joelle Thollot. Programmable 2d arrangements for element texture design. *ACM Trans. Graph.*, 36(4), May 2017.

[MBS+11] Ron Maharik, Mikhail Bessmeltsev, Alla Sheffer, Ariel Shamir, and Nathan Carr. Digital micrography. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 100:1–100:12, New York, NY, USA, 2011. ACM.

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007.

[SKA18] Reza Adhitya Saputra, Craig S. Kaplan, and Paul Asente. RepulsionPak: Deformation-driven element packing with repulsion forces. In *Proceedings of the 44th Graphics Interface Conference*, GI '18. Canadian Human-Computer Communications Society, 2018.

[SKAM17] Reza Adhitya Saputra, Craig S. Kaplan, Paul Asente, and Radomír Měch. FLOWPAK: Flow-based ornamental element packing. In *Proceedings of the 43rd Graphics Interface Conference*, GI '17, pages 8–15. Canadian Human-

Computer Communications Society, 2017.

[SLK05] Kaleigh Smith, Yunjun Liu, and Allison Klein. Animosaics. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 201–208, New York, NY, USA, 2005. ACM.

[WKSM12] B. Wyvill, P. G. Kry, R. Seidel, and D. Mould. Determining an aesthetic inscribed curve. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, CAe '12, pages 63–70, Goslar Germany, Germany, 2012. Eurographics Association.

[WZS98] Michael T. Wong, Douglas E. Zongker, and David H. Salesin. Computer-generated floral ornament. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 423–434, New York, NY, USA, 1998. ACM.

[XK07] Jie Xu and Craig S. Kaplan. Calligraphic packing. In *Proceedings of Graphics Interface 2007*, GI '07, pages 43–50, New York, NY, USA, 2007. ACM.

[ZCR+16] Changqing Zou, Junjie Cao, Warunika Ranaweera, Ibraheem Alhashim, Ping Tan, Alla Sheffer, and Hao Zhang. Legible compact calligrams. *ACM Trans. Graph.*, 35(4):122:1–122:12, July 2016.

[ZCT16] Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. Designing structurally-sound ornamental curve networks. *ACM Trans. Graph.*, 35(4):99:1–99:10, July 2016.