RepulsionPak: Deformation-Driven Element Packing with Repulsion Forces

Reza Adhitya Saputra* University of Waterloo
Craig S. Kaplan† University of Waterloo
Paul Asente‡ Adobe Research

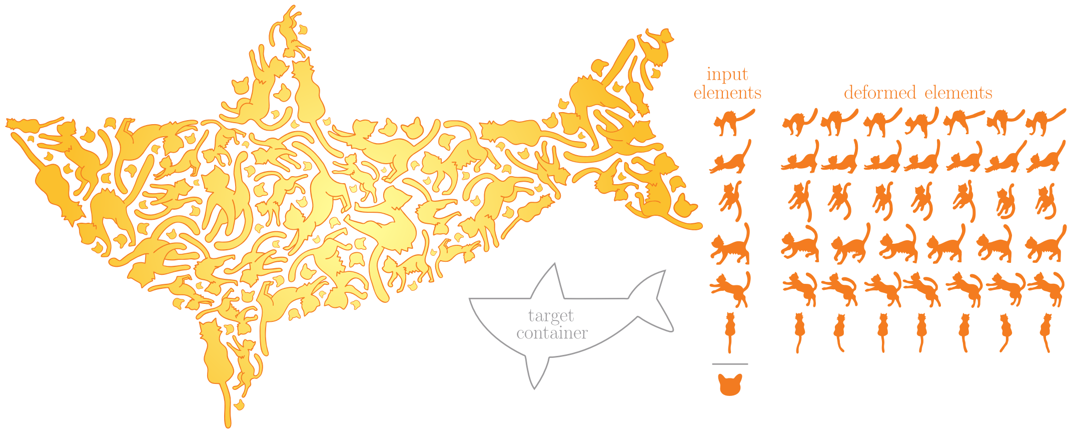

Figure 1: A packing of six cat elements inside a fish-shaped target container. Controllable deformation and repulsion forces allow the elements to deform, efficiently filling the container and creating a uniform distribution of negative space. We then reduce the remaining negative space by placing smaller cat heads. The gradient fill was added as a post-process.

Abstract

We present a method to fill a container shape with deformable instances of geometric elements selected from a library, creating a 2D artistic composition called an element packing. Each element is represented as a mass-spring system, allowing them to deform to achieve a better fit with their neighbours and the container. We start with an initial random placement of small elements and gradually transform them using repulsion forces that trade off between the evenness of the packing and the deformations of the individual elements. Our method produces compositions in which the negative space between elements is approximately uniform in width, similar to real-world examples created by artists. We validate our approach by performing a quantitative study using spatial statistics.

Keywords: packing, shape deformation, repulsion force.

Index Terms: I.3.3 [Computing Methodologies]: Computer Graphics—Picture/Image Generation; I.3.m [Computing Methodologies]: Computer Graphics—Miscellaneous;

1 Introduction

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

Most of the elements in a packing are large real-world shapes like animals, plants, or man-made objects. We refer to these as *primary elements*. An artist distributes primary elements so that they communicate the shape of the container, while attempting as much as possible to ensure an even distribution of *negative space* (sometimes called *complementary space* [7]), the subset of the container that does not belong to any element (Fig. 2, right).

*e-mail: radhitya@uwaterloo.ca
†e-mail: csk@uwaterloo.ca
‡e-mail: asente@adobe.com

The evenness of negative space plays an important role in packings. The separation between neighbouring elements should be roughly the same everywhere. When the primary elements leave behind large pockets of negative space, the artist typically fills those pockets with small *secondary elements*, often simple abstract shapes like circles or triangles. In the limit as this element separation goes to zero, the packing turns into a *tessellation*: a set of elements that exactly fill a container with no overlaps. 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.

Past work on computer-generated packings, notably Jigsaw Image Mosaics [19] and collages based on the Pyramid of Arclength Descriptor [20], 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 to design a shape descriptor that allows this compatible element to be found efficiently. Elements typically do not fit perfectly with each other or the container boundary. These techniques suppress imperfections by deforming elements in a final post-processing step.

In this paper we present RepulsionPak, a *deformation-driven* packing technique. 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.

We also incorporate an explicit simulation phase for secondary 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.

In addition to RepulsionPak itself, we also contribute a quantitative model for evaluating packings, based on measuring the evenness of the negative space. We use spherical contact probability functions to demonstrate the benefit of deformation and to compare RepulsionPak with previous work.

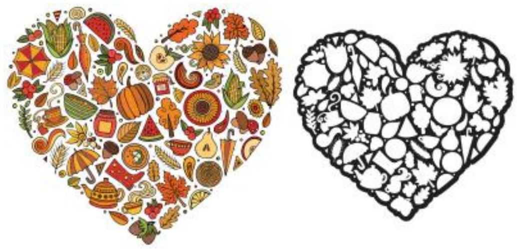

Figure 2: A packing created by hand by an artist, depicting autumn-themed elements (left), together with a visualization of its negative space (right). Small secondary elements like circles make the distribution of negative space more even. (Artist: balabolka on Shutterstock)

2 Related Work

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 [11] used a variant of Lloyd's method to pack oriented rectangles into a container region, simulating the appearance of traditional mosaics. Hiller et al. [12] extended Lloyd's method to distribute polygonal elements instead of points, reducing the overlaps in Hausner's approach. Dalal et al. [8] used an FFT-based image correlation to reposition elements iteratively, which could be seen as making more effective use of negative space, and permitting non-convex elements to interlock more than they did in earlier methods.

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. [2] and Ijiri et al. [15] use a growth model that copies small neighbourhoods from the exemplar into a larger output texture. AlMeraj et al. [1] stamp out copies of the exemplar and discard overlapping elements. Hurtut et al. [14] develop a statistical sampling method based on multitype point processes. These techniques are all concerned with replicating the uneven element distribution in the exemplar, without regard for negative space.

Dense packing and tessellations: Gal et al. [9] used local shape descriptors on 3D models to fill a 3D container with a "collage" in the style of Arcimboldo. Huang et al. [13] produced Arcimboldo-like collages in 2D by layering objects cut out from images on top of a segmented container. These methods benefit from overlaps, which join elements into a single large object. Reinert et al. [27] generated compositions by projecting objects from a high dimensional feature space down to 2D while also inferring users' intentions when manually placing elements. However, their goal was to create meaningful compositions without an attempt to effectively fill a container.

As stated in the introduction, our work is most similar to Jigsaw Image Mosaics (JIM) [19] and collages based on the Pyramid of Arclength Descriptor (PAD) [20]. JIM packed nearly-convex elements tightly by placing one element at a time and backtracking as needed. PAD developed a sophisticated shape descriptor in order to find new elements that partially matched existing element boundaries as a container was being filled. Both methods permitted some elements to overlap. While they could both correct gaps and overlaps using deformation, the deformation was applied locally near edges in a post-processing step after elements were frozen in place.

Text and letter packings: Xu and Kaplan [30] and Zou et al. [32] constructed *calligrams* by filling a container with a small number of deformed letters composing one or two words. Because the order of the letterforms was defined by the text, their solutions usually required significant distortion of the individual letters. Their goal was to balance between filling the container and preserving readability. Maharik et al. [22] explored Digital Micrography, in which whole lines of text 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 deformed ornamental elements that satisfy manufacturing constraints such as connectivity. Chen et al. [6] developed a method to synthesize filigree patterns out of simple elements. In later work, Chen et al. [5] generated modular surfaces by computing contact point networks of rigid elements.

Zehnder et al. [31] proposed an elegant method to cover 3D surfaces with deformed ornamental elastic curves. Our method has some similarities to theirs in that both start with scaled-down copies of elements and grow them, but the growth process is quite different. Unlike their approach, our elements exert forces on each other throughout the growth process, allowing them a greater opportunity to translate, rotate, and deform in search of more even negative space. Furthermore, the goal of their work (3D fabrication) is quite different from ours (2D ornamentation) and our results appear qualitatively different.

Texture atlas packing: Jiang et al. introduced the Simplicial Complex Augmentation Framework (SCAF) [18], an algorithm to create bijective maps and pack triangulated charts into a rectangular texture atlas. A SCAF begins with charts that are highly deformed to circles, and iteratively *undeforms* them, producing output charts with minimal deformation energy. RepulsionPak operates the other way around. It starts with elements that have zero deformation energy, and iteratively introduces more deformation in response to repulsion forces between elements.

Non-rigid packing: Peng et al. [26] computed layouts by packing and deforming simple polygons and polyominoes. Their method cannot handle more complicated shapes, making it unsuitable for our style of packings. FLOWPAK by Saputra et al. [28] placed ornamental elements to create a visual sense of flow. They used skeletal strokes to place elements along streamlines defined from a vector field. However, their elements could not undergo more general deformations, and their method did not explicitly control for the evenness of the negative space.

Physics-based NPR: Pedersen and Singh [25] grew curves to create organic labyrinthine paths. Their algorithm is related to ours by the use of repulsion forces to maintain even spacing and parallel segments.

3 System overview

Our system requires three main pieces of input:

1. A library of primary and optional secondary elements (Fig. 3a). Each element is a collection of open or closed polygonal paths—any curves must be flattened ahead of time.
2. One or more closed polygonal target containers, such as the heart in Fig. 3b. Target containers can optionally have internal holes.
3. The desired element spacing distance $d_{\text{gap}} > 0$.

RepulsionPak starts by preprocessing the elements, adding additional space around each to implement the spacing distance, and fitting a triangle mesh over each element. Small copies of these primary elements are randomly placed in the containers (Section 4).

It then performs a physics simulation on the meshes, making them simultaneously grow and repel each other. As a proof-of-concept, we implement a very simple spring-based simulation; many alternatives are possible (see Section 12). Forces in the simulation push mesh vertices away from vertices in other meshes, attempt to keep the meshes from undergoing excessive deformation, and resolve places where meshes overlap or vertices move outside container boundaries (Section 5).

After each iteration of the simulation, meshes grow into adjacent space if possible, so that they gradually consume the negative space in the container. At the same time, mesh self-intersections are resolved (Sections 6 and 7).

The simulation concludes when some number of sequential steps fail to significantly reduce the negative space (Section 8).

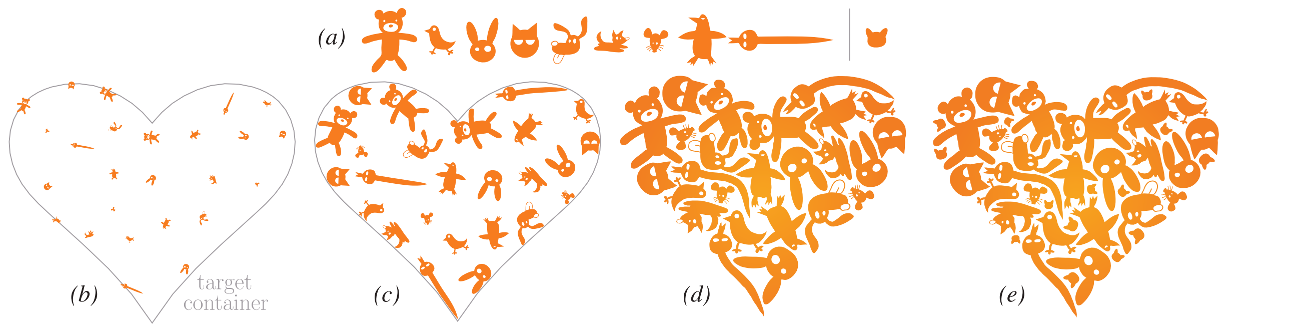

Figure 3: The creation of a packing using RepulsionPak. (a) A library of elements, comprising nine primary elements and a single secondary element. (b) A target container with the initial distribution of scaled-down elements. (c) The simulation in progress, showing the elements growing, translating, rotating, and deforming. (d) The resulting packing of primary elements. (e) The final result, after adding secondary elements and allowing them to grow. Fig. 4 shows the deformations of some of the elements.

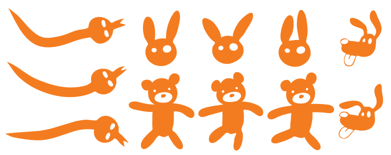

Figure 4: Some of the elements in the final packing in Fig. 3, showing the effect of deformation in our simulation.

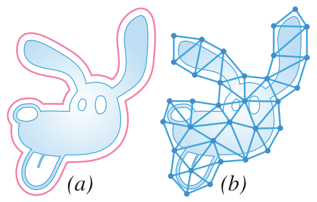

Figure 5: (a) An element with its boundary offset to create a skin, drawn in red. (b) A triangle mesh with boundary vertices on the skin.

Finally, 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 (Section 9).

Final SVG output is created by using barycentric coordinates to map each element's paths from the element's initial mesh into the deformed mesh produced through simulation.

4 Preprocessing

The *skin* of an element is a simple closed polygon that fully encloses it, as in the red shape in Figure 5(a). We generate the skin by offsetting an element's boundary outward by $d_{\text{gap}}/2$. The simulation will aim to produce an approximate tessellation of the target container by the skins, thereby achieving the desired element spacing and suppressing overlaps.

We triangulate the element skin to obtain a triangle mesh. To create the mesh we uniformly sample the skin polygon s, with samples spaced apart by distance d_{gap}, to obtain a simpler polygon s' (the outer boundary of the mesh). We then construct a Delaunay triangulation of s'. The vertices and edges of this mesh will become unit masses and longitudinal springs in a physical simulation, allowing elements to deform in response to their neighbours. We further brace the mesh against deformation by augmenting it with "auxiliary springs" (see Section 5 and Fig. 6b).

Due to discretization, a low mesh resolution does not guarantee a separation of d_{gap}. Increasing the mesh resolution will produce a more precise result at the expense of greater running time.

Barycentric coordinates: The simulation operates on meshes, not element geometry. In the final rendering phase, we will redraw an element relative to a deformed copy of its mesh. To do so, we first re-express every vertex of an element path in terms of the mesh triangles. Every element vertex lies either inside a mesh triangle or just beyond a border edge. We encode each vertex in barycentric coordinates relative to its enclosing or nearest triangle.

Initial element placement: We prepare our simulation by randomly placing non-overlapping elements.

1. Generate random points $P = \{p_1, p_2, ..., p_n\}$ inside the target container via blue noise sampling [4]. The user controls the number of points; using more points gives results with smaller elements. We can automatically estimate n by dividing the container area by the desired average area of the element skins.
2. Cycle through the primary elements, assigning each element to a random unused p_i with a random orientation, repeating until every point has an element.
3. Shrink all the elements so that they do not overlap and occupy only a small fraction of the container's area; in our implementation we have found that $5-10\%$ of the area gives good results. Making them larger would speed up the simulation but does not allow enough freedom of movement to generate successful packings. Fig. 3b shows an initial placement.

5 Forces

We design a simulation in which we generate pseudo-physical forces that transform elements by transforming their meshes. Let $\boldsymbol{x}$ be a vertex of an element mesh. The total force $\boldsymbol{F}$ applied to $\boldsymbol{x}$ is

$$\boldsymbol{F} = \boldsymbol{F}_r + \boldsymbol{F}_e + \boldsymbol{F}_b + \boldsymbol{F}_o \tag{1}$$

where $\boldsymbol{F}_r$, $\boldsymbol{F}_e$, $\boldsymbol{F}_b$, and $\boldsymbol{F}_o$ are the repulsion force, the edge force, the boundary force, and the overlap force. These forces combine with the growth process, described in Section 6, to completely fill the target container.

Repulsion force: The repulsion force tries to push element meshes apart when they approach each other, with the goal of causing them to rotate and align their boundaries (Fig. 6a).

The vertex $\boldsymbol{x}$ will experience an inverse square repulsive force, inspired by Coulomb's law, from all nearby meshes. We use the following formula:

$$\boldsymbol{F}_r = k_r \sum_{i=1}^{n} \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|} \frac{1}{\varsigma + \|\boldsymbol{u}\|^2} \tag{2}$$

where

k_r	is the strength of the repulsion force relative to other forces in the simulation.
n	is the number of nearest neighbouring meshes to $\boldsymbol{x}$.
$\boldsymbol{x}_i$	is the closest point on the skin of the ith neighbour.
$\boldsymbol{u}$	$= \boldsymbol{x} - \boldsymbol{x}_i$
ς	is a *soft parameter*; it places an upper bound on the magnitude of $\boldsymbol{F}_r$, avoiding explosive instability when $\|\boldsymbol{u}\|$ is very small.

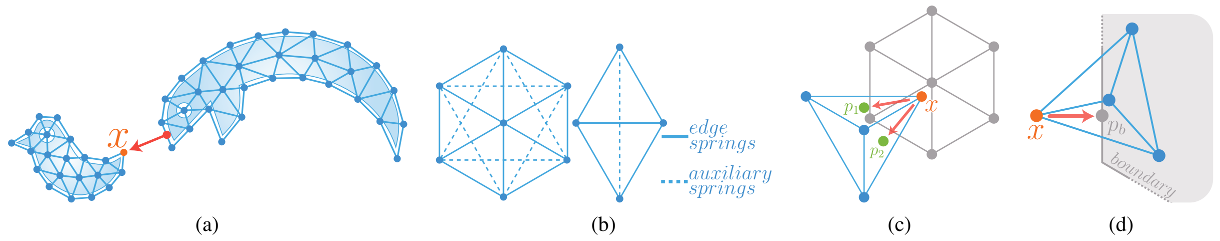

Figure 6: Illustrations of the forces in our simulation. **(a) Repulsion force**: The closest point on the snake's mesh repels a vertex $\boldsymbol{x}$ in the bird mesh. **(b) Edge force**: We generate edge forces using edge springs and auxiliary springs. **(c) Overlap force**: Centres of triangles $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ attract a vertex $\boldsymbol{x}$ that lies in the interior of another mesh. **(d) Boundary force**: A vertex $\boldsymbol{x}$ moves toward $\boldsymbol{p_b}$, the closest point on a container boundary, when it is outside the container.

An imbalance in the repulsion forces across a mesh's vertices will naturally induce translation and rotation in meshes, helping their boundaries discover compatible segments and consume more of the remaining negative space.

If $\boldsymbol{x}$ lies inside of another element's mesh, then the aggregate repulsion force from other neighbours can push $\boldsymbol{x}$ further inside and worsen the overlap. If we discover such an overlap, we set $\boldsymbol{F}_r$ to 0 and use the overlap force $\boldsymbol{F}_o$, discussed below, to correct it.

Edge force: A mesh's edges are treated as longitudinal *edge springs*; displacement of these springs allows a mesh to deform in response to repulsion forces by neighbouring meshes. In addition, when two mesh triangles share an edge we connect the two vertices opposite the edge with an *auxiliary spring* (Fig. 6b), adding extra bracing to a mesh to prevent it from folding.

An undeformed element mesh provides the rest lengths for all of its springs; as mesh vertices move relative to each other, the springs attempt to restore these rest lengths. Let $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ be mesh vertices connected by a spring. We compute the spring force as follows:

$$\boldsymbol{F}_e = k_e \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|} s\left(\|\boldsymbol{u}\| - \ell\right)^2 \tag{3}$$

where

k_e is the strength of the edge force relative to other forces.
$\boldsymbol{u} = \boldsymbol{x}_b - \boldsymbol{x}_a$
ℓ is the rest length of the spring.
s is +1 or -1, according to whether $(\|\boldsymbol{u}\| - \ell)$ is positive or negative.

We apply $\boldsymbol{F}_e$ to $\boldsymbol{x}_a$ and $-\boldsymbol{F}_e$ to $\boldsymbol{x}_b$. The equation is a modification of Hooke's law in which the strength of the force increases quadratically with displacement. This change allows the meshes to resist severe deformations when subjected to powerful forces.

Overlap force: Occasionally, a vertex $\boldsymbol{x}$ from one mesh can be pushed inside the skin of a neighbouring mesh. In such cases, we temporarily disable the repulsion force on this vertex by setting it to 0, and instead apply an overlap force that attempts to eject the intruding vertex. In particular, every mesh triangle having $\boldsymbol{x}$ as a vertex will pull $\boldsymbol{x}$ in the direction of its centroid. The overlap force is thus given by:

$$\boldsymbol{F}_o = k_o \sum_{i=1}^{n} (\boldsymbol{p}_i - \boldsymbol{x}) \tag{4}$$

where

k_o is the relative strength of the overlap force.
n is the number of mesh triangles that have $\boldsymbol{x}$ as a vertex.
$\boldsymbol{p}_i$ is the centroid of the ith triangle incident on $\boldsymbol{x}$.

The overlap force is zero for vertices that are not within another mesh.

Boundary force: The boundary force causes element meshes to stay inside the target container and conform to its boundary. It applies to any vertex $\boldsymbol{x}$ that is outside the container, and moves the vertex towards the closest point on the container's boundary, by an amount proportional to the distance to the boundary:

$$\boldsymbol{F}_b = k_b(\boldsymbol{p_b} - \boldsymbol{x}) \tag{5}$$

where

k_b is the relative strength of the boundary force.
$\boldsymbol{p_b}$ is the closest point on the container boundary to $\boldsymbol{x}$.

The boundary force is zero for any point inside the container.

Simulation details: We use explicit Euler integration to simulate the motion of the mesh vertices under the forces described above. Every vertex has a position and a velocity vector; in every iteration, we update velocities using forces, and update positions using velocities. These updates are scaled by a time step Δt, typically chosen from the range $[0.01, 0.1]$. A smaller time step results in a more stable simulation at the cost of additional running time. We cap velocities at $5\Delta t$ to dissipate extra energy from the system.

The repulsion and overlap forces rely on nearest-neighbour queries on the set of all vertices. We accelerate these queries by storing vertices in a uniform spatial subdivision grid that covers the container. In our implementation, cell width and height are $6-10\%$ of the larger dimension of the grid. We define the neighbours of a vertex $\boldsymbol{x}$ as all vertices in a 3×3 window of cells centred on the cell containing $\boldsymbol{x}$. This approximation ignores the negligible interactions between distant vertices.

The constants k_r, k_o, k_b and k_e control the relative strengths of the four forces in the simulation. They must also be chosen relative to the time step Δt and the overall width and height of the container. We find that our simulation produces satisfactory results when $k_r \approx k_o \approx k_b \geq k_e$. For example, if the container's bounding box is approximately 1000×1000, then we have obtained good results when $k_r = k_o = k_b = 80, k_e = 40$. We also set $\varsigma = 1$ to avoid explosive repulsion forces. Increasing k_e relative to the other forces suppresses deformation, yielding a close approximation of packing with rigid elements.

6 Element Growth

RepulsionPak starts with small initial elements to avoid intersections, and gradually enlarges them until they tightly fill the target container. Fig. 3c-d shows elements growing and gradually consuming negative space. Elements have different intrinsic sizes, which are respected in the initial placement. Because they all grow at roughly the same rate, their relative sizes tend to be maintained.

After each iteration of the physics simulation, the element meshes undergo a growth step. If an element mesh has no vertices that lie inside of neighbouring meshes, it is permitted to grow in this iteration. A mesh with overlaps may still grow in subsequent iterations, if local changes to the packing open up more negative space. This approach produces slight variations in skin offsets in the output packing but the effect is negligible.

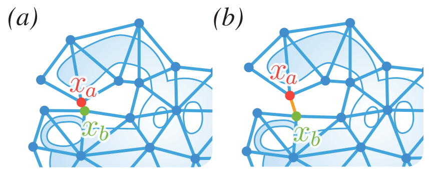

Figure 7: In (a), vertex $\boldsymbol{x}_a$ on the dog's ear is approaching $\boldsymbol{x}_b$ near the nose, threatening a self-intersection. We move the two points away from each other in (b), avoiding the intersection.

We implement growth in the context of the physics simulation by scaling the rest lengths of a mesh's springs, allowing it to expand as the simulation progresses. Every mesh M has a counter n_M that records the number of times it has grown. When a mesh is permitted to grow, we add 1 to the counter. Then, if L_i is the rest length of the ith spring in the original undeformed mesh, we increase its rest length to $(1+n_M k_g \Delta t)L_i$. The constant k_g, usually 0.01 in our system, controls the growth rate.

7 Resolving Self-Intersections

The overlap force described in Section 5 prevents overlaps between neighbouring elements, but it cannot prevent self-intersections. Fig. 7 shows an example in which an ear of a dog approaches the nose, threatening to produce a self-intersection. We adapt a relaxation method [17] to enforce the constraint that any pair of non-neighboring vertices in an element's mesh cannot be closer together than half of the average edge length for that mesh. In a separate phase, after the end of every simulation step, we calculate the average length $\bar{\ell}$ of all of the edge springs in a given mesh. If the distance between two mesh vertices is less than $\bar{\ell}/2$, our algorithm translates them away from each other until they are separated by a distance of $\bar{\ell}/2$. Resolving the constraint for a pair of vertices can invalidate it for others so we cycle through all pairs in a random order and fix all violations. We find that performing two or three cycles works well in practice.

8 Stopping Criterion

We stop the simulation and growth process when the element meshes are no longer able to maneuver enough to consume the remaining negative space. After each iteration, we compute an *area fraction* A, defined to be the fraction of the container area taken up by element meshes. We then compute a measurement of the recent change in area fraction in a sliding window that covers the w most recent iterations of the system; we use $w = 100$. If $A_0, \ldots, A_w$ are the area fractions in the $w+1$ iterations up to the current one, then we define

$$\mathrm{RMS} = \sqrt{\frac{1}{w}\sum_{i=1}^{w}(A_i - A_{i-1})^2} \tag{6}$$

We stop iterating when RMS $< \varepsilon$, where ε is 0.01 in our system.

9 Secondary Elements

The iteration process described above can leave behind isolated pockets of empty space, which will be visible in the final composition. We imitate the approach taken by human artists by filling these pockets will small, usually simple secondary elements.

We seed the container with secondary elements by finding points that are far from any existing element mesh. Specifically, we compute a discrete approximation of the distance transform of the negative space. We then create an initial candidate list of all points whose distance value is above a threshold $d_{\min}$, sorted by decreasing distance. We consider each of these candidates in turn, adding it to a final list of seed locations provided that no previously chosen seed is within distance d_{sep} of the candidate. In our implementation, if the distance transform is computed on a 1000×1000 grid fit to the container's bounds, then we typically set $5 \le d_{\min} \le 10$ and $d_{\text{sep}} = 10$.

Next, we assign random secondary elements to these chosen seed points, scaled down as before to avoid overlaps. We then run the simulation and growth process again, but freeze the primary elements: they exert repulsion forces on secondary elements and can induce overlaps, but primary mesh vertices cannot move. The secondary elements gradually grow to consume some of the remaining negative space until the packing satisfies the same stopping criterion described above.

Packings with secondary elements are shown throughout the paper; see Figures 1, 3d, and 8.

Table 1: Data and statistics for the results in the paper. The table shows the numbers of primary and secondary elements (n_p, n_s), the number of vertices (v), the running times of the primary and the secondary simulations (t_p, t_s) in seconds, and the number of iterations (i).

Packing	n_p	n_s	v	t_p	t_s	i
Cats (Fig. 1)	41	69	3598	185	62	8531
Animals (Fig. 3)	25	14	2412	133	65	16670
Birds (Fig. 8)	43	43	2309	102	54	11571
Bats (Fig. 8)	47	22	3048	165	56	13120
Butterflies (Fig. 8)	123	135	11916	696	616	14379
Collage B (Fig. 11b)	51	0	1730	121	0	14062
Collage C (Fig. 11d)	60	0	2048	341	0	31348
Autumn (Fig. 12)	82	31	4380	233	30	16890

10 Implementation and Results

Our software was written in C++, and reads in text files describing elements and containers; we prepared these files using Adobe Illustrator. We ran our software on a computer with a 2.4 GHz Intel i7-4700HQ processor and 16 GB of RAM. As a post-process, we optionally read packings back into Illustrator, fit smooth curves to polygonal paths, and applied colours and other visual effects. Table 1 shows statistics for the results in the paper. All results in this paper use $\Delta t = 0.1$.

The supplemental materials include movies that visualize the simulation process. These movies make it clear that elements can jostle each other around, inducing translation, rotation, and deformation throughout the simulation. Qualitatively, the resulting packings interlock and leave behind an even distribution of negative space.

The packing in Fig. 1 uses six cat-shaped primary elements and one secondary cat head. RepulsionPak naturally bends legs and tails to fill the container more evenly.

The animal packing in Fig. 3 has several elements with limbs (the bear, fox, chick, and penguin), extensions (the dog and bunny ears), and long shapes (the snake). Fig. 4 highlights the deformations for some of these elements; they are noticeably more deformed than nearly-convex elements like the cat and mouse.

The butterfly packing in Fig. 8 is an attempt to reproduce the visual style of a dense packing (or tessellation), similar to Jigsaw Image Mosaics [19] or the "Butterflies in Butterfly" example from the Pyramid of Arclength Descriptor paper [20, Fig. 21]. The target container is made from two regions, one with internal holes. The resulting packing is tight but overlap-free.

The packing on the left in Fig. 8 exhibits significant deformation in the wings and the tails of the birds. In particular, the thin tails of the swallows have some unaesthetic sharp bends. We would like to investigate ways to ensure these bends are smoother.

11 Evaluation

Subjectively, we believe that our results meet the aesthetic goals we defined for this style of packing. But we are also particularly interested in investigating quantitative measurements of packing quality that can be used to evaluate our results and subsequent work. We are particularly interested in statistical measurements of the

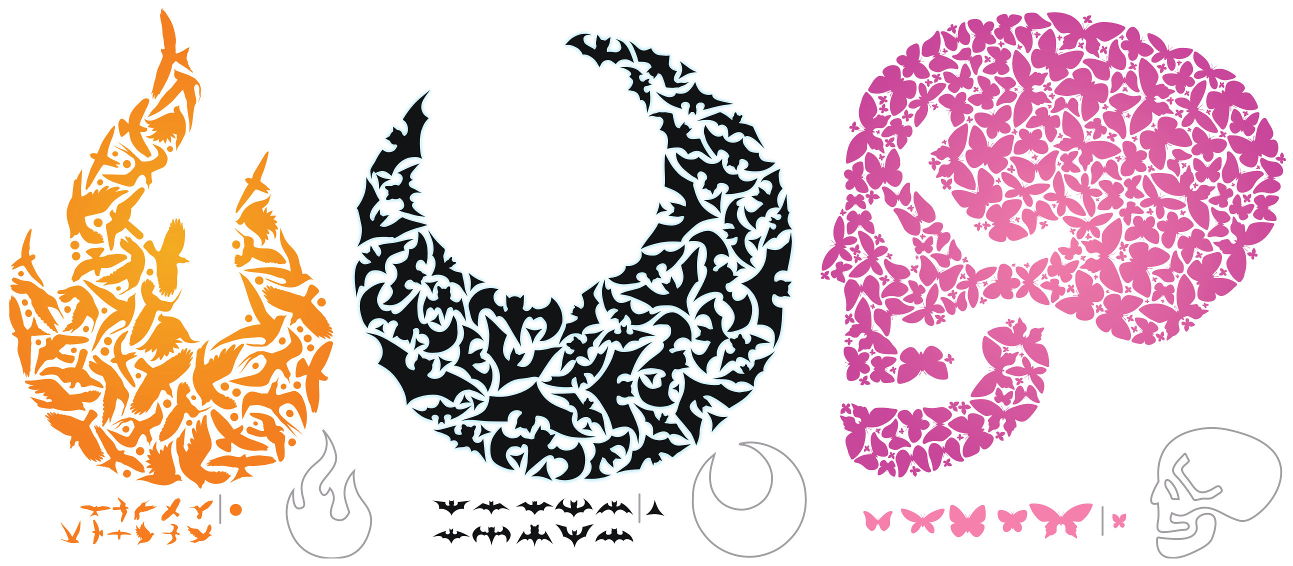
Figure 8: Three packings created using RepulsionPak: Birds, Bats, and Butterflies.

evenness of negative space, which allows us to compare our results against those of related techniques.

To evaluate the evenness of a packing's negative space, we use the *spherical contact probability* (SCP), denoted as $Q_s(r)$, which is the probability that a ball (in two dimensions, 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* [7]:

$$H_s(r) = 1 - \frac{Q_s(r)}{Q_s(0)} \tag{7}$$

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. 9a). Consider a pattern of infinite horizontal stripes of width d_s, separated from each other by d_{gap}. For this pattern, $Q_s(0) = d_{\text{gap}}/(d_{\text{gap}} + d_s)$; it is also 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 ball). Furthermore, $Q_s(r)$ will decrease linearly between these two points, 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. 9b), the intersections of horizontal and vertical channels produce pockets of negative space that can accommodate balls of radius $d_{\text{gap}}\sqrt{2}/2$. These pockets tend to raise the SCP slightly everywhere, and cause it to bend into a small tail that approaches zero gradually. For a given set of elements in a container, the best packings will have an SCP that stays close to the idealized stripe function most of the way down, has a low value at $r = d_{\text{gap}}/2$, and then bends towards horizontal near that value. In less effective packings (Fig. 9c,d), the negative space will be narrower in some places and wider in others, recognizable as a shallower SCP with a longer tail.

In this paper we compute a discrete approximation of $Q_s(r)$. First, we compute the distance transform of the negative space in a 1000×1000 grid around the container. Then, for a given r, $Q_s(r)$ is the proportion of grid points whose distance value is at least r.

Comparison to rigid packings: In order to evaluate the effect of deformation on negative space, we measure the SCP under increasing values of k_e, the spring strength. Increasing k_e allows the element meshes to resist deformation, ultimately approximating a rigid packing algorithm. We created 30 packings, 10 for each of three values of k_e (10, 250, and 1000). Each packing used 23–27 elements chosen at random from a library of 56, with no secondary elements. As shown in Fig. 10, a low value of k_e led to greater deformation and a steeper SCP, suggesting that the negative space is more even. We believe the difference is qualitatively visible in the packings themselves.

Comparison to PAD: Fig. 11 compares RepulsionPak and PAD [20] using the SCP. Packing (a) is a result from the PAD paper; Packing (b) was created with RepulsionPak using the same set of elements and with the gap width d_{gap} estimated from (a). (In our packing some man-made objects, such as the wrench, jet, rocket, ambulance, and ship, were not permitted to deform, in order to enhance recognizability.) Note that the PAD packing has several overlapping elements (for example, the tooth and the horse), but white haloes around elements artfully conceal overlaps with little degradation in visual quality. Our packing avoids overlaps by design. The SCP plot in (c) shows that our packing has a lower value at $d_{\text{gap}}/2$, indicating more even negative space, and approaches zero more quickly, indicating fewer large empty areas. The additional RepulsionPak result in (d) uses a larger set of elements from the PAD paper and a smaller value of d_{gap}, demonstrating even negative space in a tighter packing. Unlike PAD, we do not limit orientations for placed elements, leading to some elements that are placed in unnatural or difficult-to-recognize orientations.

Comparison to an artist-made packing: Fig. 12 shows RepulsionPak's results packing the elements from the artist-made Fig. 2. The elements are nearly convex, and were set to have limited deformation. The graph compares the SCP of our result with that of the artist's, showing that our negative space is more even. Our result also has fewer large empty gaps, as indicated by the inclusion of fewer secondary elements. The result shows the effectiveness of the repulsion forces in successfully discovering compatibilities in the element boundaries and filling the space effectively.

12 Conclusions and Future Work

We presented RepulsionPak, a method to create packings with deformable elements. The combination of repulsion forces and controlled deformation allows RepulsionPak to discover shape compatibilities that eliminate the need for a large element library and fill the target container effectively. Our compositions have negative space between elements that is approximately uniform in width, and we validate our approach using spherical contact probability.

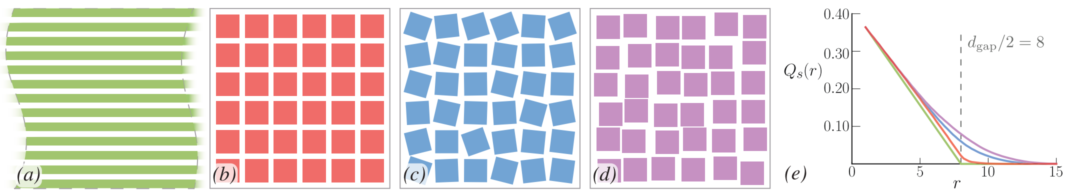

Figure 9: Spherical contact probabilities for reference packings. A "perfect packing" of infinite stripes is shown in (a), followed by a square packing with the same area fraction and d_{gap} in (b). The square packing is then perturbed with random rotations in (c) and translations in (d). The corresponding $Q_s(r)$ functions are plotted in (e).

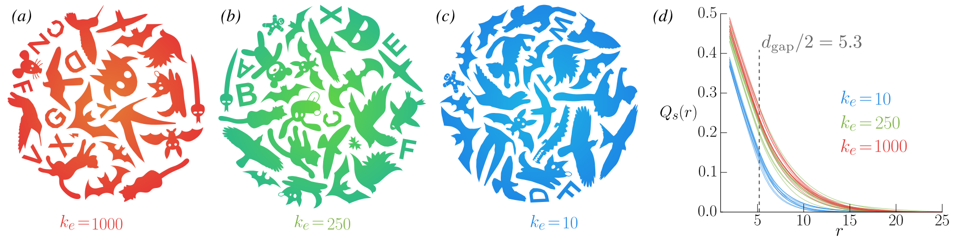

Figure 10: A demonstration of the effect of deformation on the evenness of negative space using spherical contact probability. The packings in (a), (b) and (c) are representative results using three values of the edge force strength k_e, from rigid (1000) to moderate (250) to deformable (10). We construct 10 random packings for each value of k_e, and plot their $Q_s(r)$ functions. The weakest edge force strength in (c) permits a more even distribution of negative space, as evidenced by the SCP plots.

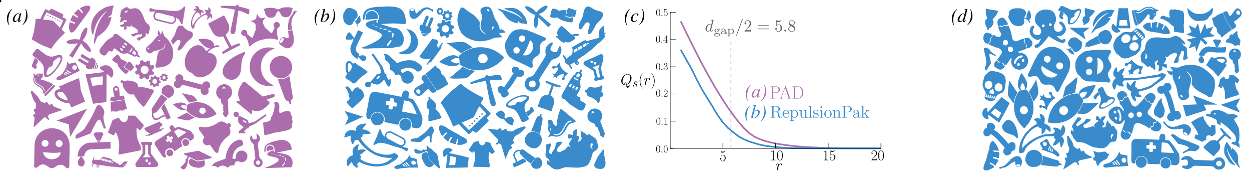

Figure 11: Packing (a) was generated by Pyramid Arclength Descriptor (PAD). Packings (b) and (d) were generated by RepulsionPak. Packing (b) has approximately the same gap width and uses the identical set of elements as packing (a). The graph (c) of $Q_s(r)$ shows that packing (b) has more even negative space with fewer large empty areas. Packing (d) is an additional RepulsionPak result with more elements and a smaller gap width.

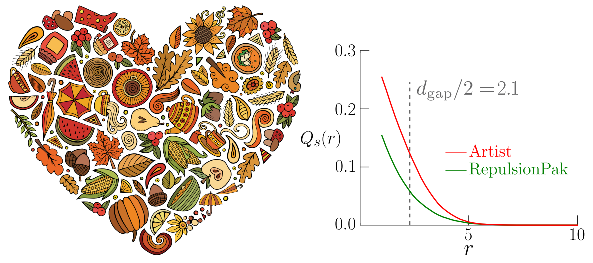

Figure 12: A packing created by RepulsionPak using the elements from Fig. 2. Our SCP is lower than the artist's result and uses fewer secondary elements. The elements are nearly convex and were set to have limited deformation; this shows the effectiveness of repulsion forces in finding compatible boundaries.

We see many possibilities for further improvements to RepulsionPak and future research on element packings.

- Because our main goal was to demonstrate the validity of a deformation-driven approach, and not to contribute a new physical simulation method, we deliberately chose a simple, open-ended simulation model based on springs and forward Euler integration. Contemporary research has yielded many more sophisticated physical simulation methods, such as Position Based Dynamics [24], Projective Dynamics [3], and the Finite Element Method. No one method is obviously best suited to this problem, and we intend to experiment with several to investigate if any offers a suitable trade-off between performance and quality.
- Shape matching techniques like PAD can produce an attractive rigid packing by finding compatibilities between neighbouring elements. It would be interesting to generate an initial element placement based on shape matching, instead of assigning elements to sample positions at random, and to investigate whether the resulting packings make even more effective use of negative space. At the very least, contour matching would allow us to place secondary elements in geometrically compatible pockets of negative space.
- 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. [31] and Gieseke et al. [10], which let the user directly manipulate elements while a composition is being created.
- As discussed in Section 11, RepulsionPak does not control for the orientations of elements, which can be problematic for shapes that become less recognizable when rotated. We would like to give the artist a way to limit element rotation. It should be possible to augment the simulation with a torsional force that attempts to restore some elements to their initial orientations.
- It would be interesting 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.
- We would like to experiment with triangulating the negative space for collision handling [18; 23]. Preventing these triangles from inverting could help to detect and resolve self-intersections and overlaps.
- Our barycentric warping method can introduce undesirable artifacts in highly deformed elements, as in the swallow tails in the left result of Fig. 8. We would like to explore other methods for warping an element's geometry based on the correspondences between the triangles of its original mesh and the deformed meshes in the final packing. To solve this distortion problem, we would like to experiment with more recent methods by Jacobson et al. [16] and Liu et al. [21].

- We would like to augment SCP with other metrics to measure how well an ornamental design fulfills other design principles. A measure of element deformation in a composition would permit a comparison against future deformation-driven techniques. At a higher level, Saputra et al. [28] argue that visual flow and "uniformity amidst variety" are important to attractive packings. In another study, Wong et al. [29] describe basic design principles for decorative arts: repetition, balance, and conformation to geometric constraints. The rigorous expression of aesthetic principles is a fascinating area for future research.

Acknowledgments

We thank the reviewers for helpful feedback. Thanks to Dietrich Stoyan for suggestions that led us to the SCP metric, Danny Kaufman for discussions about physics simulation, and Greg Philbrick for proofreading. This research was funded by the National Sciences and Engineering Research Council of Canada (NSERC) and through a generous gift from Adobe.

References

[1] Z. AlMeraj, C. S. Kaplan, and P. Asente. Patch-based geometric texture synthesis. In *Proceedings of the Symposium on Computational Aesthetics*, CAE '13, pp. 15–19. ACM, New York, NY, USA, 2013. doi: 10.1145/2487276.2487278

[2] P. Barla, S. Breslav, J. Thollot, F. Sillion, and L. Markosian. Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. of Eurographics 2006)*, vol. 25, 2006.

[3] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11, July 2014. doi: 10.1145/2601097.2601116

[4] R. Bridson. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07. ACM, New York, NY, USA, 2007. doi: 10.1145/1278780.1278807

[5] W. Chen, Y. Ma, S. Lefebvre, S. Xin, J. Martínez, and W. Wang. Fabricable tile decors. *ACM Trans. Graph.*, 36(6):175:1–175:15, Nov. 2017. doi: 10.1145/3130800.3130817

[6] W. Chen, X. Zhang, S. Xin, Y. Xia, S. Lefebvre, and W. Wang. Synthesis of filigrees for digital fabrication. *ACM Trans. Graph.*, 35(4):98:1–98:13, July 2016. doi: 10.1145/2897824.2925911

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

[8] K. Dalal, A. W. Klein, Y. Liu, and K. Smith. A spectral approach to NPR packing. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pp. 71–78. ACM, New York, NY, USA, 2006. doi: 10.1145/1124728.1124741

[9] R. Gal, O. Sorkine, T. Popa, A. Sheffer, and D. Cohen-Or. 3D collage: Expressive non-realistic modeling. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '07, pp. 7–14. ACM, New York, NY, USA, 2007. doi: 10.1145/1274871.1274873

[10] L. Gieseke, P. Asente, J. Lu, and M. Fuchs. Organized order in ornamentation. In *Proceedings of the Symposium on Computational Aesthetics*, CAE '17, pp. 4:1–4:9. ACM, New York, NY, USA, 2017. doi: 10.1145/3092912.3092913

[11] A. Hausner. Simulating decorative mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pp. 573–580. ACM, New York, NY, USA, 2001. doi: 10.1145/383259.383327

[12] S. Hiller, H. Hellwig, and O. Deussen. Beyond stippling - methods for distributing objects on the plane. *Computer Graphics Forum*, 2003. doi: 10.1111/1467-8659.00699

[13] H. Huang, L. Zhang, and H.-C. Zhang. Arcimboldo-like collage using internet images. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pp. 155:1–155:8. ACM, New York, NY, USA, 2011. doi: 10.1145/2024156.2024189

[14] 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, pp. 51–60. ACM, New York, NY, USA, 2009. doi: 10.1145/1572614.1572623

[15] T. Ijiri, R. Měch, T. Igarashi, and G. Miller. An example-based procedural system for element arrangement. *Computer Graphics Forum*, 27(2):429–436, 2008. doi: 10.1111/j.1467-8659.2008.01140.x

[16] A. Jacobson, I. Baran, J. Popović, and O. Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78:1–78:8, July 2011. doi: 10.1145/2010324.1964973

[17] T. Jakobsen. Advanced character physics. In *Proceedings of the Game Developers Conference 2001*, p. 19, 2001.

[18] Z. Jiang, S. Schaefer, and D. Panozzo. Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph.*, 36(6):186:1–186:9, Nov. 2017. doi: 10.1145/3130800.3130895

[19] J. Kim and F. Pellacini. Jigsaw image mosaics. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pp. 657–664. ACM, New York, NY, USA, 2002. doi: 10.1145/566570.566633

[20] K. C. Kwan, L. T. Sinn, C. Han, T.-T. Wong, and C.-W. Fu. Pyramid of arclength descriptor for generating collage of shapes. *ACM Trans. Graph.*, 35(6):229:1–229:12, Nov. 2016. doi: 10.1145/2980179.2980234

[21] S. Liu, A. Jacobson, and Y. Gingold. Skinning cubic Bézier splines and Catmull-Clark subdivision surfaces. *ACM Trans. Graph.*, 33(6):190:1–190:9, Nov. 2014. doi: 10.1145/2661229.2661270

[22] R. Maharik, M. Bessmeltsev, A. Sheffer, A. Shamir, and N. Carr. Digital micrography. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pp. 100:1–100:12. ACM, New York, NY, USA, 2011. doi: 10.1145/1964921.1964995

[23] M. Müller, N. Chentanez, T.-Y. Kim, and M. Macklin. Air meshes for robust collision handling. *ACM Trans. Graph.*, 34(4):133:1–133:9, July 2015. doi: 10.1145/2766907

[24] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, Apr. 2007. doi: 10.1016/j.jvcir.2007.01.005

[25] H. Pedersen and K. Singh. Organic labyrinths and mazes. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pp. 79–86. ACM, New York, NY, USA, 2006. doi: 10.1145/1124728.1124742

[26] C.-H. Peng, Y.-L. Yang, and P. Wonka. Computing layouts with deformable templates. *ACM Trans. Graph.*, 33(4):99:1–99:11, July 2014. doi: 10.1145/2601097.2601164

[27] B. Reinert, T. Ritschel, and H.-P. Seidel. Interactive by-example design of artistic packing layouts. *ACM Trans. Graph.*, 32(6):218:1–218:7, Nov. 2013. doi: 10.1145/2508363.2508409

[28] R. A. Saputra, C. S. Kaplan, P. Asente, and R. Měch. FLOWPAK: Flow-based ornamental element packing. In *Proceedings of the 43rd Graphics Interface Conference*, GI '17, pp. 8–15. Canadian Human-Computer Communications Society, 2017. doi: 10.20380/GI2017.02

[29] M. T. Wong, D. E. Zongker, and D. H. Salesin. Computer-generated floral ornament. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 423–434. ACM, New York, NY, USA, 1998. doi: 10.1145/280814.280948

[30] J. Xu and C. S. Kaplan. Calligraphic packing. In *Proceedings of Graphics Interface 2007*, GI '07, pp. 43–50. ACM, New York, NY, USA, 2007. doi: 10.1145/1268517.1268527

[31] J. Zehnder, S. Coros, and B. Thomaszewski. Designing structurally-sound ornamental curve networks. *ACM Trans. Graph.*, 35(4):99:1–99:10, July 2016. doi: 10.1145/2897824.2925888

[32] C. Zou, J. Cao, W. Ranaweera, I. Alhashim, P. Tan, A. Sheffer, and H. Zhang. Legible compact calligrams. *ACM Trans. Graph.*, 35(4):122:1–122:12, July 2016. doi: 10.1145/2897824.2925887