Improved Deformation-Driven Element Packing with RepulsionPak

Reza Adhitya Saputra, Craig S. Kaplan, and Paul Asente

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 it to deform to achieve a better fit with its neighbours and the container. We start with an initial placement of small elements and gradually transform them using a physics simulation that trades off between the evenness of the packing and the deformations of the individual elements. Unlike previous work, elements can be given preferred orientations, and we can use shape matching to control the initial placement of elements in tight convex corners. We also explore the creation of tileable packings, and validate our approach using statistical measurements of the distributions of positive and negative space in packings. Our method produces compositions in which the negative space between elements is approximately uniform in width, similar to real-world examples created by artists.

Index Terms—Element Distribution, Mosaics, Negative Space, Packing, Physical Simulation, Shape Deformation

1 Introduction

A *packing* is an arrangement of non-overlapping 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 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 shapes of real-world objects like animals, plants, or human artifacts. 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* [1]), the subset of the container that does not belong to any element (Fig. 1, right).

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 [2] and collages based on the Pyramid of Arclength Descriptor [3], can 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 target container. 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 target container. 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. We believe that the aesthetic quality of the packing as a whole more than compensates for the relatively small deformations applied to the elements.

Fig. 1. 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)

- *R. A. Saputra and C. S. Kaplan are with the Department of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada. E-mail: {radhitya, csk}@uwaterloo.ca*
- *P. Asente is with Adobe Research, San Jose, CA, 95110, United States. E-mail: asente@adobe.com*

In addition to the RepulsionPak algorithm itself, we are also interested in methods for measuring the evenness of negative space in a packing, which provides a quantitative means of evaluating and comparing packing algorithms. In this paper we discuss several possible measurements of evenness based on methods from spatial statistics.

This article includes major extensions to our previous conference paper on packing deformable elements [4]. We incorporate torsional forces to control the orientations of elements (Section 5), use a shape matching technique to improve the quality of initial element placement (Section 10), and investigate statistical measures of negative space in more detail (Section 12). In addition we offer a more robust method for resolving element self-intersections (Section 7), a new stopping criterion (Section 8), and a demonstration of tileable packings (Section 12).

2 Related Work

Rigid packing: The ever-popular Lloyd's method [5] is an iterative process for creating a perceptually even distribution of points, and has been used in various forms in procedural packing methods. Hausner [6] used a variant of Lloyd's method to pack oriented rectangles into a container region, simulating the appearance of traditional mosaics. Hiller et al. [7] 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 distributions of rigidly transformed elements that mimic the statistics of an exemplar. Barla et al. [9] and Ijiri et al. [10] use a growth model that copies small neighbourhoods from the exemplar into a larger output texture. AlMeraj et al. [11] stamp out copies of the exemplar and discard overlapping elements. Hurtut et al. [12] 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. [13] used local shape descriptors on 3D models to fill a 3D container with a "collage" in the style of Arcimboldo. Huang et al. [14] 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. [15] 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) [2] and collages based on the Pyramid of Arclength Descriptor (PAD) [3]. 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 [16] and Zou et al. [17] 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. [18] 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. [19] developed a method to synthesize filigree patterns out of simple elements. In later work, Chen et al. [20] generated modular surfaces by computing contact point networks of rigid elements.

Zehnder et al. [21] 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) [22], 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. [23] 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. [24] 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.

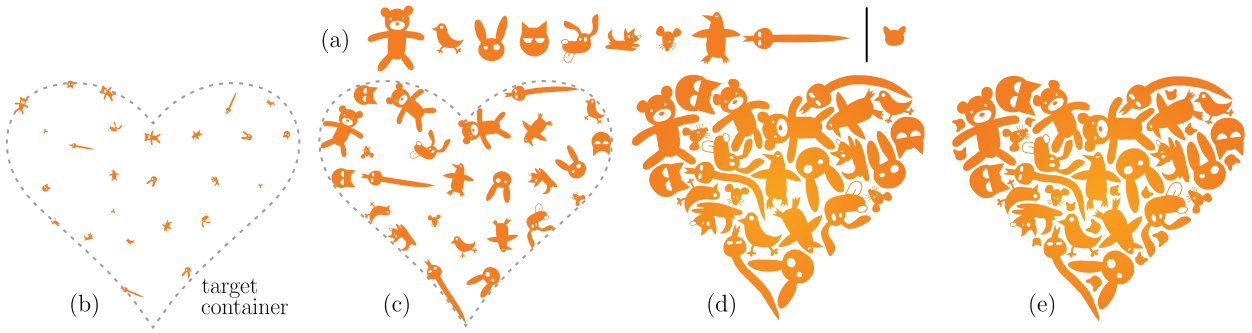

Fig. 2. The creation of a packing using RepulsionPak. (a) A library of elements, comprising nine primary elements and a single secondary element. (b) A target container with the initial distribution of scaled-down elements. (c) The simulation in progress, showing the elements growing, translating, rotating, and deforming. (d) The resulting packing of primary elements. (e) The final result, after adding secondary elements and allowing them to grow. Fig. 3 shows the deformations of some of the elements.

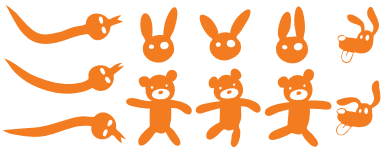

Fig. 3. Some of the elements in the final packing in Fig. 2, showing the effect of deformation in our simulation.

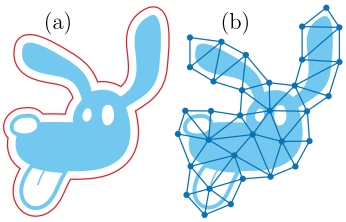

Fig. 4. An illustration of element discretization for preprocessing. (a) An element with its boundary displaced to create a skin, drawn in red. (b) A triangle mesh with boundary vertices on the skin.

3 System Overview

Our system requires three main pieces of input:

1. A library of primary and optional secondary elements, such as those shown in Fig. 2a. 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. 2b. Target containers can optionally have internal holes.
3. The desired element spacing distance $d_{\text{gap}} > 0$.

RepulsionPak starts by preprocessing the elements, creating additional space around each to enforce the spacing distance, and fitting a triangle mesh over each element. The containers are then seeded with randomly placed copies of scaled-down elements (Section 4).

RepulsionPak then performs a physics simulation on the meshes, making them simultaneously grow and repel each other. As a proof of concept, we implement a spring-based simulation, which yields satisfactory results despite its simplicity; many alternatives are possible (see Section 13). 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 target containers (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 either when the elements occupy a sufficient proportion of the container area, or when some number of simulation steps fail to significantly reduce the negative space (Section 8).

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 4(a). We generate the skin by offsetting an element's boundary outward by $d_{\text{gap}}/2$. The simulation aims to produce an approximate tessellation of the target container by deformed 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 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. 5b).

Due to discretization, a low mesh resolution does not guarantee a separation of d_{gap}. Increasing the mesh resolution produces 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.

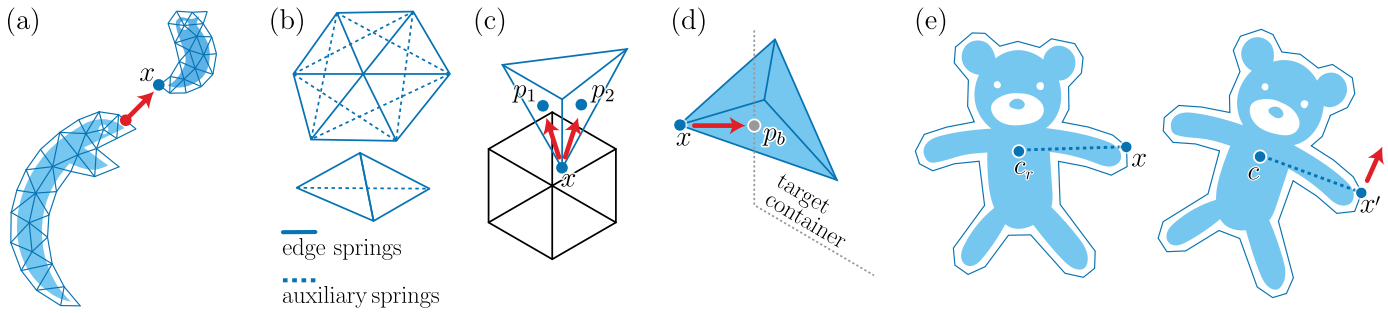

Fig. 5. Illustrations of the forces in our simulation. **(a) Repulsion force**: The closest point on the snake's mesh repels a vertex $\boldsymbol{x}$ in the bird mesh. **(b) Edge force**: We generate edge forces using edge springs and auxiliary springs. **(c) Overlap force**: Centres of triangles $\boldsymbol{p_1}$ 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 target container, when it is outside the container. **(e) Torsional force**: We restrict the orientation of the element by generating a torsional force at every boundary vertex $\boldsymbol{x}$ to restore its angular position relative to $\boldsymbol{c}$, the center of mass of the element.

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 [26]. 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. 2b 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

$$\boldsymbol{F} = \boldsymbol{F_r} + \boldsymbol{F_e} + \boldsymbol{F_b} + \boldsymbol{F_o} + \boldsymbol{F_t} \tag{1}$$

where $\boldsymbol{F_r}$, $\boldsymbol{F_e}$, $\boldsymbol{F_b}$, $\boldsymbol{F_o}$ and $\boldsymbol{F_t}$ are the repulsion force, the edge force, the boundary force, the overlap force, and the torsional 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 making them transform to align their boundaries (Fig. 5a).

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.

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 consuming 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. 5b), 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:

$$\boldsymbol{F_e} = k_e \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|} s\,(\|\boldsymbol{u}\| - \ell)^2 \tag{3}$$

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 target container to $\boldsymbol{x}$.

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

Torsional force: As forces propagate through an element mesh, the aggregate velocity vectors of the vertices can induce a rotation of the entire element. However, some elements may have a preferred orientation, either for aesthetic reasons or because the shape is comprehensible only at certain orientations. We introduce a torsional force that penalizes individual vertices for which the orientation, relative to their element's center of mass, drifts too far from its initial orientation.

Consider a vertex $\boldsymbol{x}$ belonging to an element, and let $\boldsymbol{c}_r$ be the element's center of mass in its undeformed state. We may define the "rest orientation" of $\boldsymbol{x}$ as the orientation of the vector $\boldsymbol{u}_r = \boldsymbol{x} - \boldsymbol{c}_r$. During simulation we compute the current centre of mass $\boldsymbol{c}$ of the element and let $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{c}$. Then the torsional force is

$$\boldsymbol{F}_t = \begin{cases} k_t \boldsymbol{u}^{\perp}, & \theta > 0 \\ -k_t \boldsymbol{u}^{\perp}, & \theta < 0 \end{cases} \tag{6}$$

where

k_t is the relative strength of the torsional force.
θ is the signed angle between $\boldsymbol{u}_r$ and $\boldsymbol{u}$;
$\boldsymbol{u}^{\perp}$ is a unit vector rotated $90°$ counterclockwise relative to $\boldsymbol{u}$; and

Using the equation above, $\boldsymbol{F}_t$ is always perpendicular to $\boldsymbol{u}$ and the direction of $\boldsymbol{F}_t$ points to the left or to the right depending on θ. Unlike the first four force types, the torsional force is optional.

Simulation details: We use explicit Euler integration to simulate the motions 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 k_e, and k_t control the relative strengths of the five 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 $k_r \approx k_o \approx k_b \geq k_e > k_t$. For example, if the container's bounding box is approximately 1000×1000, then we have obtained good results when $k_r = k_o = k_b = 80$, $k_e = 40$, and $k_t = 1$. 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. 2c-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.

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 that occur when distant parts of the same

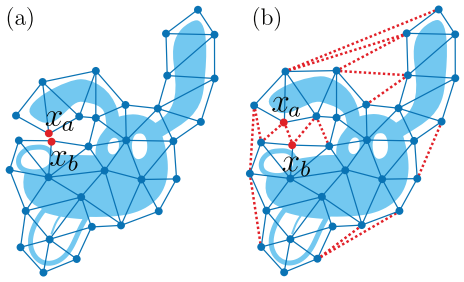

Fig. 6. An illustration of resolving self-intersection using negative space springs. In (a), vertex $\boldsymbol{x_a}$ on the dog's ear is approaching $\boldsymbol{x_b}$ near the nose, threatening a self-intersection. We triangulate the negative space inside the convex hull to create negative space springs (drawn as dashed red lines) that generate forces to prevent the intersection.

element bend until they come into contact. Inspired by Air Meshes [27], we triangulate the negative space inside the convex hull of the element and obtain additional *negative space springs* (Fig. 6). A negative space spring generates a force only if its length is shorter than 10% of the average of edge lengths of an element; otherwise it stays inactive, allowing the element more freedom to deform. In our implementation, we set the strength of negative space springs to be 50% of that of the edge springs. Unlike Air Meshes, we triangulate only the negative space inside each element's convex hull, rather than all negative space in the packing. This simplification avoids costly global re-triangulation as elements shift.

8 Stopping Criteria

We choose one of two criteria to stop the simulation. First, the artist specifies the desired positive space ratio at which the simulation immediately terminates. The ratio should not be set too high, and we find that most packings have positive space ratios between 45% and 65% depending on the element shapes. For example, concave elements with long extensions are more difficult to pack, so a lower positive space ratio is recommended to generate a satisfactory packing. Second, we stop the simulation 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

$$\mathrm{RMS} = \sqrt{\frac{1}{w}\sum_{i=1}^{w}(A_i - A_{i-1})^2} \tag{7}$$

We stop iterating when $\mathrm{RMS} < \epsilon$, 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 for which the 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 $d_{\mathrm{sep}} = 10$.

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 criteria described above.

Packings with secondary elements are shown throughout the paper; see Figs. 2e, 9, and 10.

10 Shape Matching

The motions and deformations of elements, as described in the previous sections, give them an opportunity to conform to each other and to the target container. However, in some cases the random seeding may position some elements in such a way that the simulation process will still leave undesirable artifacts. In particular, when a round element is placed near a sharp convex corner of the container, it cannot deform enough to extend into the corner, but cannot yield its position to a pointy element that offers a better fit. In the final packing, the sharp corners of the containers will appear "rounded off". Another problem occurs when a long narrow element is initially placed diagonally across a corner, in which case the simulation pushes the element's middle into the corner, causing extreme deformation.

To overcome these deficiencies, we optionally perform an initial fit-guided placement pass before seeding the rest of the container at random. Here we take inspiration from existing rigid packing algorithms [3], which are driven primarily by shape compatibility. We use a simplified shape descriptor; we can tolerate a less perfect initial fit, with the expectation that deformation will improve the quality later.

We begin by building shape descriptors for the elements. Each element is first scaled to have area $0.6A_c/n_e$, where A_c is the container area and n_e is the number of elements. This step resizes the element to a rough estimate of its final size in the packing, an approximation that is adequate for our fit-based placement. We then sample the target container and the boundaries of the elements, with adjacent samples separated by distance δ. We set $\delta = 0.002L_s$, where L_s is the side length of the container's bounding square.

We define a local descriptor based on integral of absolute curvature [3], [28]. Let $P(t)$ be an arclength-parameterized 2D curve, and let $\kappa(t)$ represent the curvature of the curve

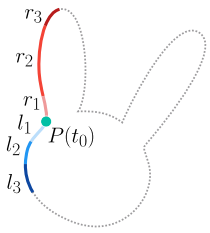

Fig. 7. An illustration of a local shape descriptor with $n = 3$. These segments have varying arclengths but they all have the same value of τ, the integral of absolute curvature along their lengths.

at $P(t)$. For a given interval $[s, t]$ within the curve's domain, we may define the integral of absolute curvature $\tau(s,t) = \int_s^t |\kappa(x)|dx$. In practice, we estimate curvature from the discrete sample points using second-order forward finite differences [29] and compute τ by summing curvature estimates using the trapezoid rule.

Let an objective τ_{obj} be a positive real number and $P(t_0)$ be a given point on a curve. If τ_{obj} is sufficiently small, then as we traverse the curve on either side of $P(t_0)$ we will eventually reach points $P(t_{-1})$ and $P(t_1)$ such that $\tau(t_{-1}, t_0) = \tau(t_0, t_1) = \tau_{\text{obj}}$. We let $l_1 = t_0 - t_{-1}$ and $r_1 = t_1 - t_0$ be the arclengths that produce these integrals. We can continue this process for any number of steps, walking in both directions along the curve away from previous sample points until we reach τ_{obj}, yielding new arclengths l_k and r_k (see Fig. 7). Finally, for a given number of steps n we define the shape descriptor at $P(t_0)$ as $(l_n, l_{n-1}, \ldots, l_2, l_1, r_1, r_2, \ldots, r_{n-1}, r_n)$. Like PAD [3], our shape descriptor is rotation invariant. Effectively it is one level of a PAD, which suffices because we do not require scale invariance. Descriptors can be compared using simple Euclidean distance, accelerated by storing them in a k-D tree. In our implementation we set $\tau_{\text{obj}} = 0.001L_s$; the dependence on L_s makes the measurement robust against changes in absolute container size, and the factor of 0.001 was determined through experimentation. We further choose $n = 5$, yielding a 10-dimensional descriptor. We compute this descriptor for all the container and element samples defined above.

Based on these descriptors, we now use a simple greedy heuristic to identify salient container features where shape matching will be used. Here we restrict our attention to convex protrusions with high curvature, which benefit the most from careful element placement (See Fig. 8a). Given a shape descriptor, we define its total length to be $\sum_{i=1}^{n} l_i + r_i$. We iterate over all sample points on the container boundary in increasing order by the total lengths of their descriptors. For each sample point $P(t_0)$, we add it to a list of salient features under two conditions. First, we require that the angle formed by $P(t_0)$ with two samples to either side of it be sufficiently acute; we use a threshold of 0.3 radians to ensure convex-to-convex matching. Second, we ensure that salient features are not too close to each other by requiring that every new sample point be separated by a distance of at least $0.2L_s$ from all previously identified salient features.

Even when an element's descriptor is a close match to a segment of target container, it may still not be safe to place that element at a given location. For example, one part of an element may extend into a corner of the container, while a different part, outside the purview of the local shape descriptor, could protrude outside the container entirely. We augment our descriptor-based fit calculation with an additional score adapted from Gal et al. [13] in order to ensure a more global element fit. Let $d(\boldsymbol{x})$ be a signed distance function for the container, with negative values inside the container and positive outside. In practice we superimpose a 1000×1000 grid over the container's bounding square and compute a discrete approximation of the distance transform. For an element sample point $\boldsymbol{x}$, we then compute a score

$$q(\boldsymbol{x}) = \begin{cases} e^{-d(\boldsymbol{x})^2/\mu^2\beta}, & \text{if } d(\boldsymbol{x}) < 0 \\ 1, & \text{if } d(\boldsymbol{x}) = 0 \\ -\alpha d(\boldsymbol{x}), & \text{if } d(\boldsymbol{x}) > 0 \end{cases} \tag{8}$$

With this scoring function, a sample point that lies on the container boundary will have a score of 1, the maximum for $q(\boldsymbol{x})$. Scores decay exponentially towards 0 for points farther inside the container. Sample points outside the container are penalized by assigning a negative score proportional to distance.

We then define a quality measurement for an entire element by summing over all of its sample points:

$$Q = \sum_{i=1}^{n_p} q(\boldsymbol{x}_i) \tag{9}$$

The parameter α penalizes elements with parts that protrude outside the container; β favors elements with parts close to the target container; $\mu = 0.5\sqrt{2}L_s$ is half of the diagonal of the container's bounding square; and n_p is the number of sample points on the element. In our implementation we choose $\alpha = 1$ and $\beta = 0.001$. For every salient container feature, we obtain the 10 closest descriptors from the k-D tree, and choose one that has the highest Q. Finally, we place the selected element in the container by aligning the endpoints of the element's and container's matching shape descriptors.

The shape matching process yields a set of elements that should be attached to particular locations on the target

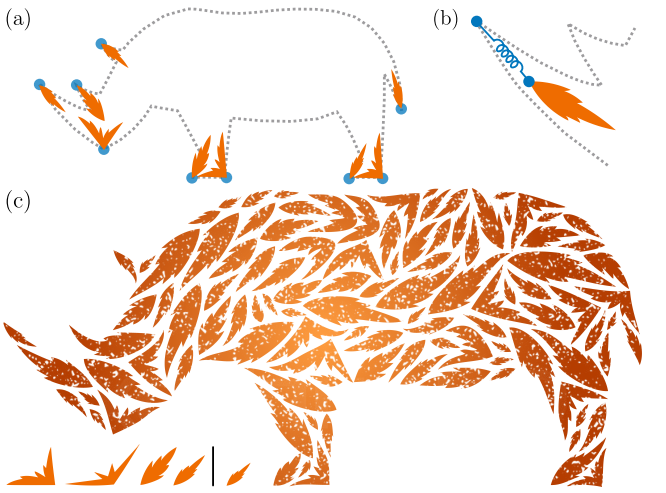

Fig. 8. A demonstration of shape matching of leaf shapes inside a rhinoceros. (a) We detect nine salient features, namely sharp convex corners, and assign an element to each. (b) A spring holds each element in place. (c) The final result.

container. However, during simulation they could wander away from these initial positions, under the influence of the many other forces at play. As shown in Fig. 8b, we encourage these elements to remain in place by attaching them to the salient container points with additional springs.

11 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 apply colours and other visual effects. Table 1 shows statistics for the results in the paper. All results in this paper use $\Delta t = 0.1$. Torsional forces are used only in Fig. 11, and shape matching is used only in Fig. 8. We also use our improved method for resolving self intersection in Figs. 8, 11, 12, 16, and 17.

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.

The packing in Fig. 9 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. 2 has several elements with limbs (the bear, fox, chick, and penguin), extensions (the dog and bunny ears), and long shapes (the snake). Fig. 3 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. 10 is an attempt to reproduce the visual style of a dense packing (or tessellation), similar to Jigsaw Image Mosaics [2] or the "Butterflies in Butterfly" example from the Pyramid of Arclength Descriptor paper [3, 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. 10 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.

The packing in Fig. 11 demonstrates torsional forces that gradually turn the elements upside down. The rhinoceros packing in Fig. 8 demonstrates initial placement via shape matching. The entire shape matching process took 922 milliseconds with a 4-element library.

We have also experimented with creating tileable packings, as shown in Fig. 12. We seed a central square with elements, but also place clones of those elements in all the squares of the 8-neighbourhood around the center. These clones track the transformations and deformations of their originals, but also exert forces on them during simulation, leading to an even, seamless packing in a toroidal domain.

We have found that RepulsionPak is robust to parameter variation, and produces predictable, high quality results without the need for fine-grained adjustments. However, as shown in Fig. 13, extreme parameter settings can still produce degenerate results. In Fig. 13a, the repulsion force is made much stronger than the edge force, leading to excessive element deformation and self-intersections in the pursuit of even negative space. In Fig. 13b, edge and torsional forces dominate repulsion, producing a packing with stiff, upright elements that do not fill their container effectively.

12 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 believe that perceived packing quality is closely tied to the evenness of negative space, a claim that we hope to evaluate more fully in future work. Accordingly, we have developed several measurements of evenness, which allow us to examine the behaviour of manually constructed reference packings and to compare RepulsionPak with other packing algorithms.

Two packings must be calibrated to each other before our measurements can be compared meaningfully. We can compare packings of different sizes by normalizing their containers to have unit area. We must also arrange for the packings to have the same *negative space ratio* (the overall amount of negative space as a fraction of container area), so that our measurements can focus on the distribution of negative space and not just the amount. Given two calibrated packings, we examine three main evenness metrics: overlap of offset elements, spherical contact probability functions, and distance histograms. We discuss each of these metrics in detail below.

The **overlap function** is a function of a non-negative offset amount r. For any given r, we offset every element by computing its Minkowski sum with a disc of radius r. As r grows, elements will start overlapping; the overlap function measures the total area of these overlaps, normalized by container area, as a function of r. We can also visualize these overlapping areas directly as in Fig. 14. In a perfect packing, we would expect no overlaps until $r = d_{\mathrm{gap}}/2$, our desired gap distance, at which point overlapping areas would start to grow into channels of roughly even width.

The **spherical contact probability** (SCP) 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 [1]. The SCP can be summarized via a function $Q_s(r)$ that gives this probability for each radius r. In order to interpret the SCP, 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 d_{gap}. For this pattern, $Q_s(0) = d_{\mathrm{gap}}/(d_{\mathrm{gap}} + d_{\mathrm{s}})$; it is also clear that $Q_s(d_{\mathrm{gap}}/2) = 0$, 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. 15b), the intersections of horizontal and vertical channels produce pockets of negative space that can accommodate balls of radius $d_{\mathrm{gap}}\sqrt{2}/2$. These pockets tend to raise the SCP slightly everywhere, and cause

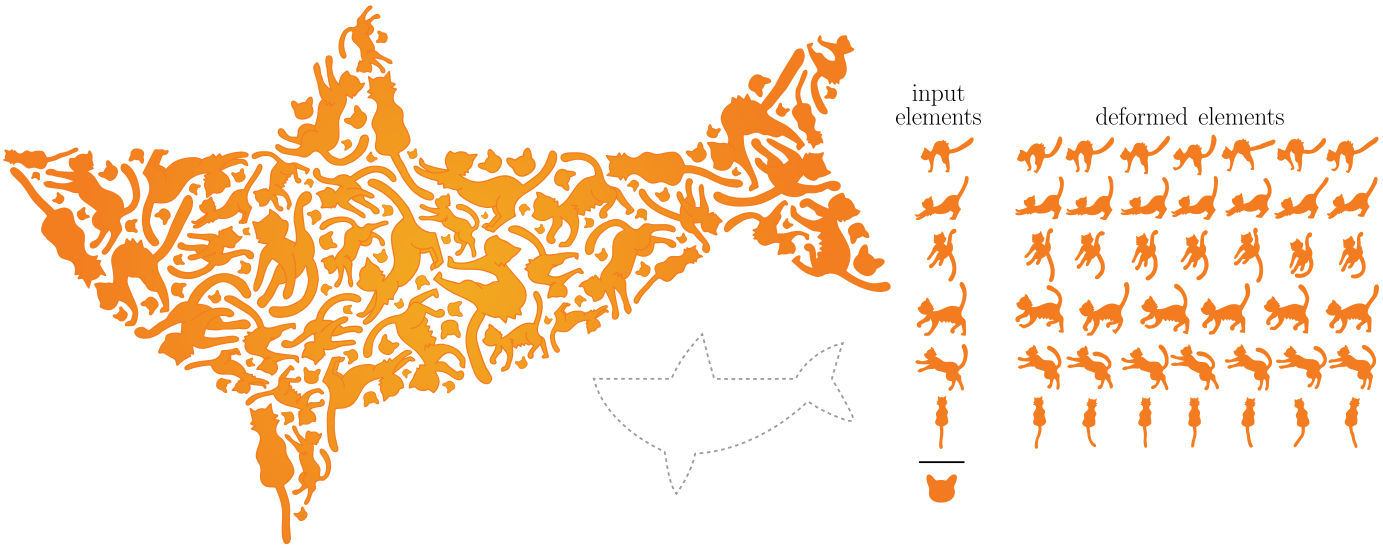

Fig. 9. A packing of six cat elements inside a fish-shaped target container. Controllable deformation and repulsion forces allow the elements to deform, efficiently filling the container and creating a uniform distribution of negative space. We then reduce the remaining negative space by placing smaller cat heads. The gradient fill was added as a post-process.

Fig. 10. Three packings created using RepulsionPak: Birds, Bats, and Butterflies. The results are visually appealing overall, though some birds' tails suffer from excessive deformation in the packing on the left.

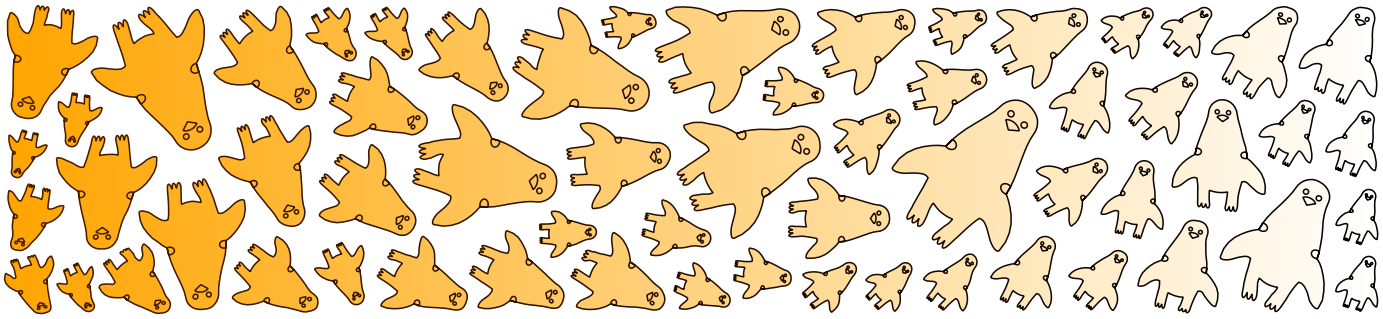

Fig. 11. A packing that demonstrates torsional forces. The packing uses copies of a single element shape, but every copy is given a rest orientation between 0° and 180°, based on its horizontal position in the container. In the final packing the elements transition from upright to upside-down, recreating an illusion in which giraffe heads become penguins.

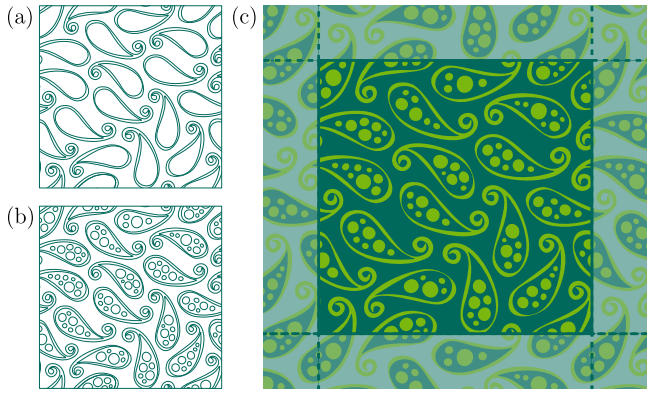

Fig. 12. A paisley-inspired toroidal packing that can tile the plane. (a) An initial paisley packing. (b) In a separate simulation, we fill each paisley with circles to demonstrate a packing inside a packing. (c) The final result.

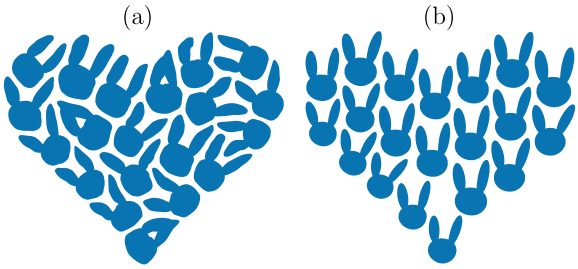

Fig. 13. Two demonstrations of how extreme parameter values can lead to low-quality results. In (a), we allow repulsion to overwhelm element shape by setting k_r to 200 and k_e to 20; the resulting packing has even negative space, but elements suffer from high deformation and self intersections. In (b), we minimize repulsion and prioritize orientation by setting k_e, k_t, and k_r to 200, 100, and 50, respectively. The elements deviate minimally from their original shapes and orientations, but cannot fill the container effectively.

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). In these results, torsional forces are used only in Fig. 11 and shape matching is used only in Fig. 8.

Packing	n_p	n_s	v	t_p	t_s	i
Animals (Fig. 2)	25	14	2412	133	65	16670
Rhino (Fig. 8)	107	0	4833	237	0	15521
Cats (Fig. 9)	41	69	3598	185	62	8531
Birds (Fig. 10a)	43	43	2309	102	54	11571
Bats (Fig. 10b)	47	22	3048	165	56	13120
Butterflies (Fig. 10c)	123	135	11916	696	616	14379
Giraffes & Penguins (Fig. 11)	60	0	2250	163	0	9347
Paisley (Fig. 12)	162	0	4860	128	0	23040
Circles in Paisleys (Fig. 12)	144	0	2544	403	0	29584
Collage (Fig. 16)	51	0	3481	729	0	29868
Autumn (Fig. 17)	77	0	2427	868	0	38387

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 a steeply-decreasing 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. Note that there is always a largest disc that can fit in a packing's negative space, and hence a largest r for which $Q_s(r) > 0$. In our graphs, we plot $Q_s(r)$ only until this point, allowing us to compare the largest empty gaps of two packings. In less effective packings (Figs. 15c,d), the negative space will be narrower in some places and wider in others, recognizable as a shallower SCP with a longer tail.

Saputra et al. [4] calculated a discrete approximation of the SCP. We can compute it more accurately geometrically, by offsetting the negative space inward. Let N be the shape of the negative space (essentially the container region with holes where elements are). For a given radius r, we compute $N(r)$, the Minkowski difference of N with a disc of radius r. Then $Q_s(r)$ is simply the ratio of the area of $N(r)$ to the area of the container.

The distance transform of the negative space can also provide insights into how negative space varies. We could calculate a standard **histogram of the distance transform**, but that would require quantizing distances into bins. Instead, note that the SCP $Q_s(r)$ is precisely the normalized area of negative space for which the distance transform is at least r. Looked at another way, if we offset each element by a radius r, then $1 - Q_s(r)$ must be the normalized area of the union of these offset elements. This area can then be interpreted as a cumulative distribution function of distance. From this observation we can compute a continuous variant of the distance histogram as a probability density function via the derivative of the SCP: $H(r) = -Q_s'(r)$.

Given two calibrated packings, the areas under their distance histograms are the same. But a more even packing will have a shorter tail, indicating a tighter upper bound on gap size, and it will have a larger concentration of density around $d_{\text{gap}}/2$. In Fig. 15, the histogram for the perfect stripe pattern is a step function that drops to zero at $d_{\text{gap}}/2$. The ideal square packing has a histogram that climbs gently until around $d_{\text{gap}}/2$ before dropping steeply. The other two packings have shallower, smoother histograms. Note also that high values of the distance histogram near $d_{\text{gap}}/2$ correspond to a rapid negative change in the SCP, suggesting a more even packing.

Comparison to PAD: Fig. 16 compares RepulsionPak and PAD [3]. Packing (a) is a result from the PAD paper; Packing (b) was created with RepulsionPak using the same elements, and calibrated to have the same negative space as (a). Note that the PAD packing actually 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 has a shorter tail, indicating fewer large empty areas. Our result also has a histogram bump around $d_{\text{gap}}/2$, and a lower overlap function.

Comparison to an artist-made packing: In Fig. 17 we show a RepulsionPak result created using the elements from the artist-made packing in Fig. 1. Our packing was calibrated to match the artist's. Looking closely, the artist's packing has a few elements separated by narrow gaps, such as the cherries and the corn on the top left. Our result has fewer large empty gaps, as indicated by a short tail in its SCP. Our result also has a histogram bump around $d_{\text{gap}}/2$, and a lower overlap function. The result shows the effectiveness of the repulsion forces in successfully discovering compatibilities in the element boundaries and filling the space effectively.

Comparison to rigid packings: To evaluate the effect of deformation on negative space, we compute all three metrics under increasing values of k_e, the edge spring strength. In-

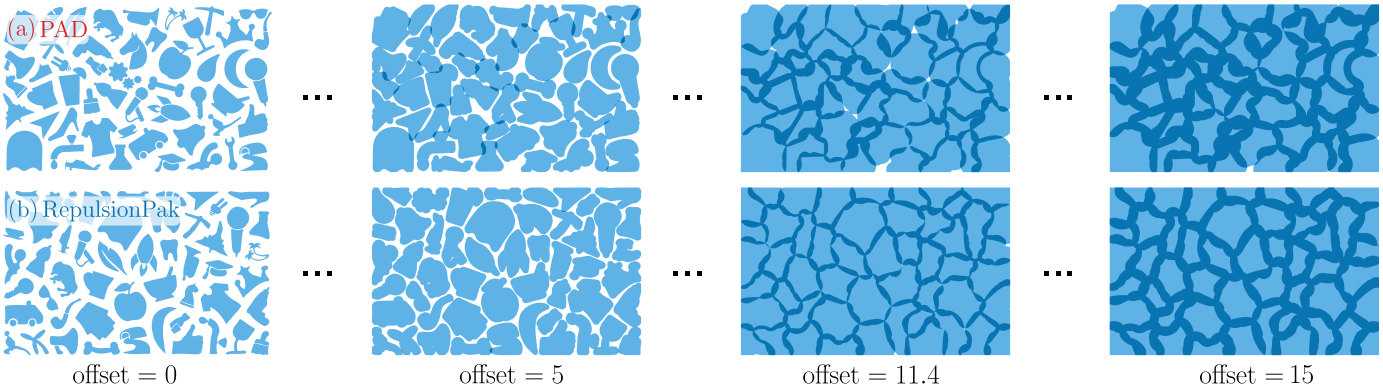

Fig. 14. An illustration of offsetting elements outward. The packings have $d_{\text{gap}}/2 = 5.7$. At an offset of 5, which is slightly less than $d_{\text{gap}}/2$, the overlap for the PAD packing is 1.462% of the total area, while the overlap for our packing is only 0.039%. At an offset of 11.4, which equals d_{gap}, the PAD packing shows more empty space (1.05%) than RepulsionPak (0.07%). As the offset is increased, overlaps in the PAD packing create channels with uneven widths, whereas ours are more uniform.

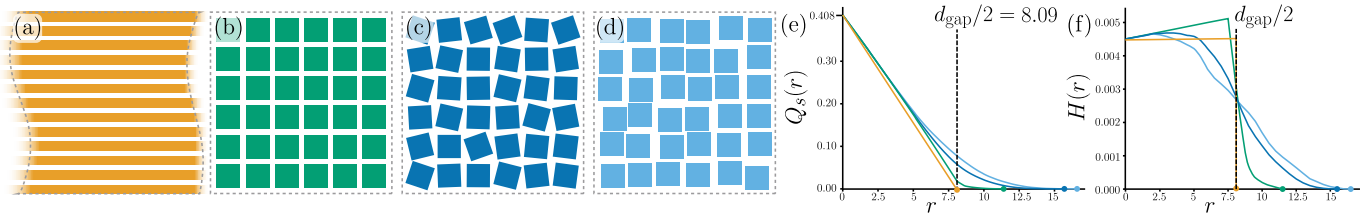

Fig. 15. Spherical contact probabilities and distance histograms for reference packings. A "perfect packing" of infinite stripes is shown in (a), followed by a square packing with the same area fraction and d_{gap} in (b). The square packing is then perturbed with random rotations in (c) and translations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

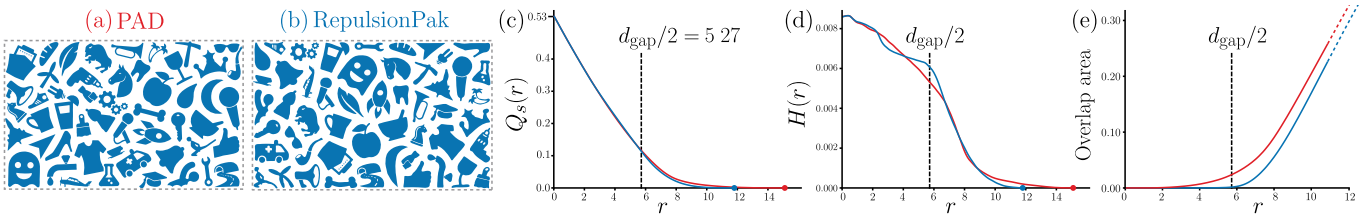

Fig. 16. A comparison between a PAD packing shown in (a) and a RepulsionPak packing in (b) with their corresponding SCPs (c), distance histograms (d), and overlap functions (e). The PAD and RepulsionPak packings are calibrated to have the same negative space ratio. Our SCP is lower and shorter than the PAD's result, our histogram shows higher concentration around $d_{\text{gap}}/2$, indicating more even negative space, and our packing has a lower overlap function.

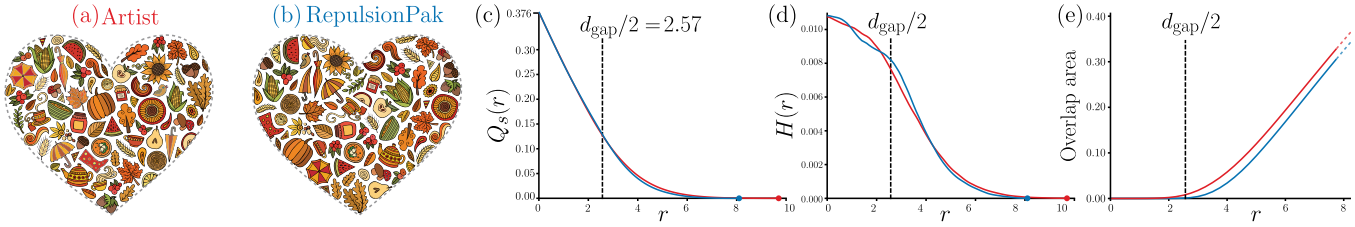

Fig. 17. A comparison between the artist-made packing from Fig. 1, shown in (a), and a RepulsionPak packing with the same elements in (b). We plot the corresponding SCPs (c), distance histograms (d), and overlap functions (e). For comparison purposes we remove secondary elements from the artist's packing. Our SCP is lower and shorter than the artist's result, our histogram shows more concentration around $d_{\text{gap}}/2$, and RepulsionPak also has a lower overlap function.

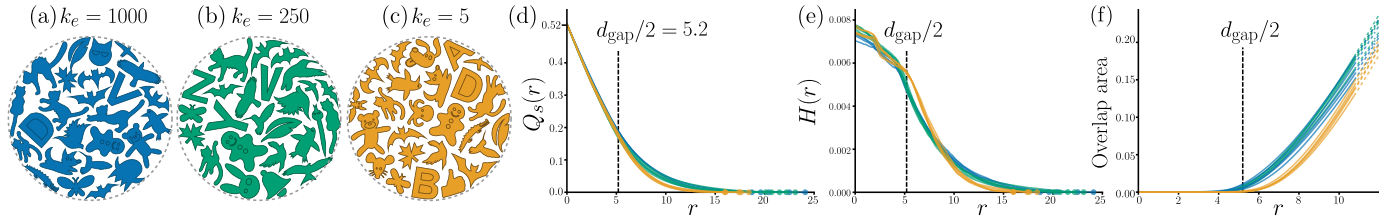

Fig. 18. A demonstration of the effect of deformation on the evenness of negative space. The packings in (a), (b) and (c) are representative results using three values of the edge force strength k_e, from rigid ($k_e = 1000$) to moderate ($k_e = 250$) to deformable ($k_e = 5$). We construct five random packings for each value of k_e, and plot their SCPs (d), histograms (e), and overlap functions (f). Packings with the most deformation have steeper and shorter SCPs, more histogram concentrations around $d_{\text{gap}}/2$, and lower overlap functions.

creasing k_e allows the element meshes to resist deformation, ultimately approximating a rigid packing algorithm. We created 15 packings, five for each of three values of k_e (5, 250, and 1000). Each packing used 25 elements chosen at random from a library of 60, with no secondary elements. All packings are partly calibrated: they all have the same negative space ratio, but elements are chosen at random and have some variation in their sizes. As shown in Fig. 18, a low value of k_e leads to greater deformation and steeper SCPs. The more pronounced histogram bump at $r = d_{\text{gap}}/2$ suggests more even negative space. The lower overlap functions indicate fewer narrow gaps between elements.

13 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 overlap functions, spherical contact probability functions, and distance histograms.

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 simulation model based on springs and forward Euler integration. Contemporary research has yielded many more sophisticated physical simulation methods, such as Position Based Dynamics [30], Projective Dynamics [31], 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.
- We would like to explore the use of RepulsionPak in a fabrication context. For example, our boundary compatibilities might be used 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.
- Our barycentric warping method can introduce undesirable artifacts in highly deformed elements, as in the swallow tails in the left result of Fig. 10. 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, based for example on the work of Jacobson et al. [32] and Liu et al. [33].
- We would like to conduct experiments that investigate the extent to which quantitative measurements of the evenness of negative space in a packing correlate with the human perception of a packing's quality. In informal evaluations, some viewers found that the packing in Fig. 17b, created with RepulsionPak, was packed more tightly than the artist's packing in Fig. 17a, even though both have the same total amount of negative space.
- Our validation metrics are all based on Minkowski sums or differences with discs, corresponding to a form of shape offsetting where corners become round. The result of this rounding is visible in our graphs, for example in the gradual flattening of the SCP for the squares in Fig. 15e. It would be worthwhile to repeat these measurements using mitered offsetting, and to evaluate whether rounded or mitered offsetting is a closer match to human perceptual judgment of evenness.
- When comparing calibrated packings, SCPs communicate differences in the evenness of negative space, but the differences between SCPs can be subtle. In addition to distance histograms, we would like to investigate other visualizations of this information that might amplify these differences to make evaluation easier.
- We would like to develop additional metrics to evaluate 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. [24] argue that visual flow and "uniformity amidst variety" are important to attractive packings. In another study, Wong et al. [34] 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 reviewers for helpful feedback. Thanks to Dietrich Stoyan for discussions about evaluation metrics, Danny Kaufman for discussions about physics simulation, and Greg Philbrick for providing feedback on an earlier draft. This research was funded by the National Sciences and Engineering Research Council of Canada (NSERC) and through a generous gift from Adobe.

References

[1] S. Chiu, D. Stoyan, W. Kendall, and J. Mecke, *Stochastic Geometry and Its Applications*, ser. Wiley Series in Probability and Statistics. Wiley, 2013. [Online]. Available: https://books.google.ca/books?id=GCRI8Q-RUEkC

[2] J. Kim and F. Pellacini, "Jigsaw image mosaics," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 657–664. [Online]. Available: http://doi.acm.org/10.1145/566570.566633

[3] 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.*, vol. 35, no. 6, pp. 229:1–229:12, Nov. 2016. [Online]. Available: http://doi.acm.org/10.1145/2980179.2980234

[4] R. Saputra, C. Kaplan, and P. Asente, "RepulsionPak: Deformation-driven element packing with repulsion forces," in *Proceedings of Graphics Interface 2018*, ser. GI 2018. Canadian Human-Computer Communications Society, 2018, pp. 10 – 17.

[5] M. McCool and E. Fiume, "Hierarchical poisson disk sampling distributions," in *Proceedings of the Conference on Graphics Interface '92*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 94–105. [Online]. Available: http://dl.acm.org/citation.cfm?id=155294.155306

[6] A. Hausner, "Simulating decorative mosaics," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 573–580. [Online]. Available: http://doi.acm.org/10.1145/383259.383327

[7] S. Hiller, H. Hellwig, and O. Deussen, "Beyond stippling—methods for distributing objects on the plane," *Computer Graphics Forum*, vol. 22, no. 3, pp. 515–522, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00699

[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*, ser. NPAR '06. New York, NY, USA: ACM, 2006, pp. 71–78. [Online]. Available: http://doi.acm.org/10.1145/1124728.1124741

[9] 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. [Online]. Available: http://maverick.inria.fr/Publications/2006/BBTSM06

[10] T. Ijiri, R. Měch, T. Igarashi, and G. Miller, "An example-based procedural system for element arrangement," *Computer Graphics Forum*, vol. 27, no. 2, pp. 429–436, 2008. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2008.01140.x

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

[12] 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*, ser. NPAR '09. New York, NY, USA: ACM, 2009, pp. 51–60. [Online]. Available: http://doi.acm.org/10.1145/1572614.1572623

[13] 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*, ser. NPAR '07. New York, NY, USA: ACM, 2007, pp. 7–14. [Online]. Available: http://doi.acm.org/10.1145/1274871.1274873

[14] H. Huang, L. Zhang, and H.-C. Zhang, "Arcimboldo-like collage using internet images," in *SIGGRAPH Asia '11*, ser. SA '11. New York, NY, USA: ACM, 2011, pp. 155:1–155:8. [Online]. Available: http://doi.acm.org/10.1145/2024156.2024189

[15] B. Reinert, T. Ritschel, and H.-P. Seidel, "Interactive by-example design of artistic packing layouts," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 218:1–218:7, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508363.2508409

[16] J. Xu and C. S. Kaplan, "Calligraphic packing," in *Proceedings of Graphics Interface 2007*, ser. GI '07. New York, NY, USA: ACM, 2007, pp. 43–50. [Online]. Available: http://doi.acm.org/10.1145/1268517.1268527

[17] C. Zou, J. Cao, W. Ranaweera, I. Alhashim, P. Tan, A. Sheffer, and H. Zhang, "Legible compact calligrams," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 122:1–122:12, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925887

[18] R. Maharik, M. Bessmeltsev, A. Sheffer, A. Shamir, and N. Carr, "Digital micrography," in *SIGGRAPH '11*, ser. SIGGRAPH '11. New York, NY, USA: ACM, 2011, pp. 100:1–100:12. [Online]. Available: http://doi.acm.org/10.1145/1964921.1964995

[19] W. Chen, X. Zhang, S. Xin, Y. Xia, S. Lefebvre, and W. Wang, "Synthesis of filigrees for digital fabrication," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 98:1–98:13, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925911

[20] W. Chen, Y. Ma, S. Lefebvre, S. Xin, J. Martínez, and W. Wang, "Fabricable tile decors," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 175:1–175:15, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130817

[21] J. Zehnder, S. Coros, and B. Thomaszewski, "Designing structurally-sound ornamental curve networks," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 99:1–99:10, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925888

[22] Z. Jiang, S. Schaefer, and D. Panozzo, "Simplicial complex augmentation framework for bijective maps," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 186:1–186:9, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130895

[23] C.-H. Peng, Y.-L. Yang, and P. Wonka, "Computing layouts with deformable templates," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 99:1–99:11, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2601097.2601164

[24] 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*, ser. GI '17. Canadian Human-Computer Communications Society, 2017, pp. 8–15. [Online]. Available: https://doi.org/10.20380/GI2017.02

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

[26] R. Bridson, "Fast Poisson disk sampling in arbitrary dimensions," in *ACM SIGGRAPH 2007 Sketches*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1278780.1278807

[27] M. Müller, N. Chentanez, T.-Y. Kim, and M. Macklin, "Air meshes for robust collision handling," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 133:1–133:9, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2766907

[28] M. Cui, J. Femiani, J. Hu, P. Wonka, and A. Razdan, "Curve matching for open 2D curves," *Pattern Recogn. Lett.*, vol. 30, no. 1, pp. 1–10, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2008.08.013

[29] T. F. Banchoff and S. T. Lovett, *Differential Geometry of Curves and Surfaces*. Chapman and Hall/CRC, 2015.

[30] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *J. Vis. Comun. Image Represent.*, vol. 18, no. 2, pp. 109–118, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jvcir.2007.01.005

[31] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, "Projective dynamics: Fusing constraint projections for fast simulation," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 154:1–154:11, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2601097.2601116

[32] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 78:1–78:8, Jul. 2011. [Online]. Available: http://doi.acm.org/10.1145/2010324.1964973

[33] S. Liu, A. Jacobson, and Y. Gingold, "Skinning cubic Bézier splines and Catmull-Clark subdivision surfaces," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 190:1–190:9, Nov. 2014. [Online]. Available: http://doi.acm.org/10.1145/2661229.2661270

[34] 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*, ser. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 423–434. [Online]. Available: http://doi.acm.org/10.1145/280814.280948

Reza Adhitya Saputra is a PhD Candidate in the University of Waterloo. His interests include vector graphics, non-photorealistic rendering, and art. He is a recipient of Cheriton scholarship in 2018.

Craig S. Kaplan is a Computer Science Professor at the University of Waterloo. He has a BMath in Pure Mathematics and Computer Science from Waterloo, and an MS and PhD in Computer Science from the University of Washington. He studies the application of computer graphics and mathematics to problems in art, architecture and design. He is as Associate Editor of the Journal of Mathematics and the Arts, and helps to organize the annual Bridges Conference on art and mathematics.

Paul Asente is a Senior Principal Scientist with Adobe Research. His main research areas are 2D design, vector graphics, and stylization. He received his A.B. in mathematics from Dartmouth College in 1979 and his Ph.D. in computer science from Stanford University in 1987. In the distant past he was a major contributor to the X Window System and has contributed to many Adobe products including Illustrator, Photoshop, After Effects, and Character Animator.