Dense Packing of Congruent Circles in Free-form Non-convex Containers

Jinesh Machchhar[a], Gershon Elber[a]

[a]*Department of Computer Science, Technion Israel Institute of Technology, Israel*

Abstract

This paper proposes an algorithm for computing dense packings of congruent circles inside general 2D containers. Unlike the previous approaches which accept as containers, only simple, symmetric shapes such as circles, rectangles and triangles, our method works for any container with a general, freeform (spline) boundary. In contrast to most previous approaches which cast the problem into a non-convex optimization problem, our method attempts to maximize the number of packed circles via a perturbation approach and consists of two main phases. In the first phase, an initial packing is computed by placing circles in spiraling layers, starting along the boundary of the container. The next phase simulates the shaking of a container under gravity, thereby making room for additional circles by perturbing the existing circles. While the general circle packing problem is known to be NP-hard [13], our method proposes heuristics which lead to dense packings. Comparison of results with previous approaches on simple, symmetric shapes shows the effectiveness of our algorithm while results of packing inside freeform containers demonstrates the generality of our algorithm.

1. Introduction

This paper addresses the problem of dense packing of congruent circles inside general 2D shapes. This involves placing as many circles of a given radius as possible, inside the given shape so that no two circles overlap. This is a well-studied problem in mathematics and in general, is known to be NP-hard [13]. While proven optimal solutions are known only for certain shapes and specific radii [7], a large body of literature is devoted to proposing heuristics for computing maximally dense packings.

The circle packing problem enjoys a wide range of applications. In architecture, circle packings are employed to obtain aesthetic patterns [18]. Pottmann et al. [19] explore substitution of conventional geometric objects such as meshes in graphics and architectural geometry, with other combinatorial objects such as circles. Covering of surfaces with circle and sphere packings, with applications towards architectural geometry is discussed in [20]. Stippling is an artistic technique which achieves monochrome rendering of objects by varying the distances between the dots (stipples) placed in different regions [14]. Circle packing with different radii in different regions could potentially lead to stippling. Several industrial applications of circle packing are discussed in [3], such as the problem of cutting maximum number of circles from rectangular metal strips and cutting circular blanks for manufacturing stators and rotors of electric motors. Further, packing of cylindrical objects into rectangular cases may be reduced to packing circles in rectangular shapes. The problem of

locating a set of maximally dispersed points in a convex region for the purpose of facility location may be approached via the equivalent problem of circle packing [5]. Also, circle packings in a similar setting have been used for computing conformal maps [22].

While all the previous approaches are targeted towards densely packing circles inside simple convex shapes such as rectangles, squares, circles, triangles, etc., to the best of our knowledge, this is the first attempt at packing circles inside arbitrary non-convex shapes with free-form boundaries. In contrast to the traditional approach of posing the problem as a non-convex optimization problem, we propose a randomized algorithm which simulates the movement of balls inside a rotated container under shaking and gravity. As observed in practice, such a process is likely to create room for more items to insert. Our approach captures the interaction between the circles and the container boundary, given in freeform spline parametric form, by solving systems of algebraic constraints [11]. Notably, each of the previous methods is geared towards a specific regular shape on which it improves the previously known best results. While our method accepts any general freeform shape as input, results of comparison on simple symmetric shapes show that we almost always achieve the best known packing when the shape is rectangular.

This paper is organized as follows. A brief survey of previous methods is given in Section 2. In Section 3, we state the basic notation and definitions and give an overview of our two-phase approach. The first phase is an initialization step, which produces a sub-optimal packing by placing circles in layers and is explained in Section 4. The second phase, which involves perturbation of existing circles, is described in Section 5 and aims to iteratively improve the packing by moving all the circles in randomly chosen directions, possibly creating room for inserting additional circles. Results from an implementation of our algorithm using the IRIT [6] modeling kernel are given in Section 6. We conclude the paper in Section 7 with remarks on future directions.

2. Previous work

Given the large volume of literature, it is beyond the scope of this paper to give a thorough survey of the existing methods for circle packing. Instead, here we present a brief overview of some of the representative previous approaches. A comprehensive survey of circle-packing algorithms is given by Hifi and M'Hallah [12]. One of the earliest attempts at packing circles inside a circular container was due to Kravitz [15] in 1967. A prominent approach to packing congruent circles inside circular or square shaped containers is to pose it as a non-convex optimization problem. Graham et al. [9] proposed the method of optimization by repulsion forces which aimed to maximize the minimum distance between all pairs of centers of circles, thereby, maximizing the radius of circles being packed in the circular container of unit radius. Mladenovic et al. [17] propose a method to escape the stationary points encountered while solving the optimization problem, by switching from one formulation to another, using change of coordinates between Cartesian and polar. Their method is tailored towards circular containers. In 2008, Grosso et al. [10] propose an approach for packing circles inside a circular container wherein, each time a local minimum is found by the local search procedure, it is perturbed and the local search is repeated. Birgin and Sorbal [1] speed-up the computation by dividing the container into a grid so that constraints enforcing non-overlapping of circles need only to be enforced on circles in adjacent cells of the grid. Their method accepts containers with symmetric shapes such as circle, square, triangle and rectangle. Lopez and Beasley [16] work with multiple formulations as well as restrict the range of movement of centers of circles to local regions. A physics

based algorithm for packing circles inside a circular container was given by Graham and Lubachevsky [8] which simulates movement of billiards balls inside the container. The balls move along straight lines, maintaining the constraint of no overlap. A packing is obtained by allowing the balls to grow in radius, as much as possible. Chen et al. [4] give a shaking based packing algorithm which only accepts polygonal containers as input. Their method proceeds by choosing a subset of the hexagonal packing of the infinite plane that may be accommodated inside the container. The shaking step consists of pushing a single circle to corners of the polygon. Further details about the shaking step are missing in the paper.

None of the above methods consider general, freeform containers. Further, most of the previous approaches fix the number of circles to be packed and either increase the radius of the circles or shrink the container, in order to achieve a dense packing. In contrast, our method fixes the container as well as the radius of the circles, while increasing the number of circles packed. Contact with the free-form boundary of the container is handled here via solutions of algebraic constraints [11] using the multivariate solver of the IRIT [6] modeler. We use the B-spline representation for the boundary curves.

3. Overview of our approach

In this section, we formulate the problem, set up the notation, and describe the basic mathematical tools which are used throughout the paper. Given a closed, bounded, 2D container D with boundary ∂D, and radius r of circles, the goal is to find a maximal packing of circles inside D, i.e., to place as many non-overlapping circles inside D as possible. Let $d(p, \partial D)$ be the Euclidean distance between point p and the point of ∂D which is closest to p. Then, such a packing must obey the following two criteria:

Criterion 1. *The center, p, of each circle packed in D must belong to the interior, D^o, of D and $d(p, \partial D) \geq r$.*

The above criteria ensures that each circle is completely contained within D.

Criterion 2. *For each pair of circles packed in D, the distance between their centers, p_i and p_j must be at least $2r$, i.e., $d(p_i, p_j) \geq 2r$.*

The second criterion ensures that no two circles overlap. We now introduce some notation which is used in the rest of the paper.

Definition 3. For a given domain, a **configuration**, denoted by $\boldsymbol{\mathcal{C}}$, is a set of circles, each of radius r, obeying Criteria 1 and 2.

Definition 4. Given a configuration $\boldsymbol{\mathcal{C}}$, the **free-space** is defined as the set of points in D^o which lie in neither the interior nor the boundary of any circle in $\boldsymbol{\mathcal{C}}$, i.e., $\left\{ p \in D^o \cap \left(\bigcap_{C \in \boldsymbol{\mathcal{C}}} C^e \right) \right\}$, where C^e denotes the exterior of circle $C \in \boldsymbol{\mathcal{C}}$.

Because the container D is closed and bounded, the free-space is an open set and has no boundary. With a slight abuse of notation, we will refer to the boundary of the closure in D, of any connected component, F, of free-space, as the boundary of F and denote it by ∂F. Figure 1 illustrates a configuration with two circles. Two components of free-space are indicated with their boundaries and are shown in blue and green. For the ease of discussion, D is assumed to be simply connected.

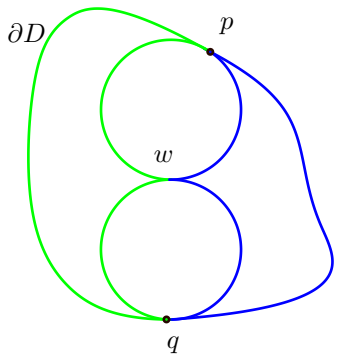


Figure 1: A configuration with two packed circles, which make contact with ∂D at points p and q. The two circles make contact at the point labeled w. The free-space has two connected components shown by their boundaries in green and blue.

The following Lemma characterizes the nature of tangential contacts made by each circle with other circles as well as the boundary of the container in $\boldsymbol{\mathcal{C}}$ and is used extensively in our approach.

Lemma 5. *Consider a simply connected 2D region, D, bound by a C^2 continuous closed curve ∂D. One of the following two cases hold for D:*

(a) If ∂D has a point with curvature, κ, greater than $\frac{1}{r}$, then exactly one of the following holds:

  (i) No circle with radius r can be packed inside D or

  (ii) A circle can be packed inside D which makes, two or more tangential contacts with ∂D.

(b) If the curvature of ∂D is bounded from above by $\frac{1}{r}$, i.e., for all $q \in \partial D$, $\kappa(q) \leq \frac{1}{r}$, then a circle can be packed inside D which makes a single contact with ∂D.

Proof.
(a) Suppose first, that there exists a point in ∂D with curvature greater than $\frac{1}{r}$. Consider the medial axis, γ, of D and let $d_\gamma : \gamma \to \mathbb{R}$ be the real function mapping each point p of γ to the distance between point p and a point $q \in \partial D$ closest to p. Since D is simply connected, γ is connected. Further, since ∂D is continuous, d_γ is continuous. Since there exists a point in ∂D with curvature greater than $\frac{1}{r}$, there exists a curvature local maximum, say at $q_0 \in \partial D$, such that $\kappa(q_0) > \frac{1}{r}$. Clearly, for the point $p_0 \in \gamma$ corresponding to q_0, $d_\gamma(p_0) = \frac{1}{\kappa(q_0)} < r$.

Suppose that a circle with radius r can be packed inside D. This implies that there exists a point p_1 in γ such that $d_\gamma(p_1) \geq r$. By connectedness of γ and continuity of d_γ it follows that there exists a point p_2 in γ such that $d_\gamma(p_2) = r$. It follows that, in general, the circle with center p_2 and radius r makes two or more tangential contacts with ∂D. This is illustrated schematically in Figure 2 wherein, the medial axis of D is shown in magenta. The circle shown in red, with center p_0, corresponds to the point in ∂D with curvature local maximum. The circles shown in blue and green have radius greater than r and equal to r, respectively.

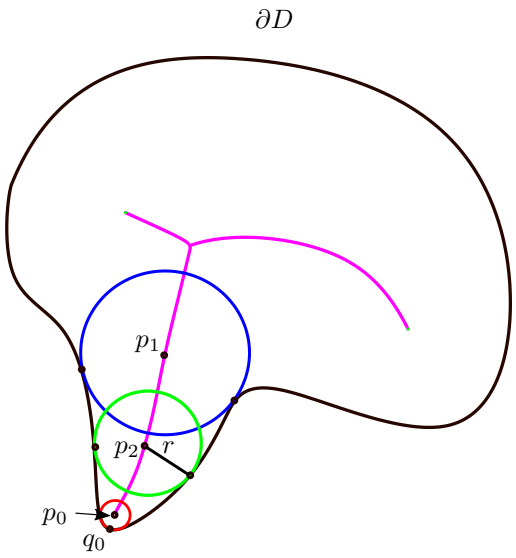


Figure 2: The medial-axis of ∂D is shown in magenta. The circle shown in red, with center p_0 and radius less than r, corresponds to the point, q_0, of curvature local maximum in ∂D. The circle shown in blue, with center p_1 has radius greater than r. The circle shown in green with center p_2 has radius equal to r.

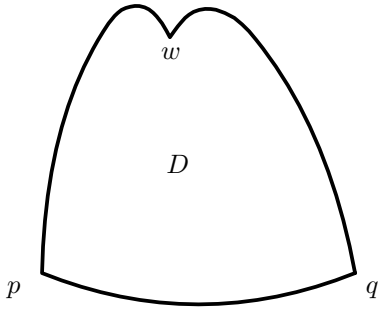


Figure 3: The boundary ∂D is C^2-discontinuous at points p, q and w. The domain D is locally convex at p, q while it is locally concave at w.

(b) Suppose now that for all $q \in \partial D$, $\kappa(q) \leq \frac{1}{r}$. By an argument similar to that given in the previous case, it follows that $\min_{p\in\gamma} d_\gamma(p) \geq r$. Hence, it is possible to place a circle of radius r inside D making a single contact with ∂D. □

Note that in case *(b)*, when $\kappa(q) = \frac{1}{r}$, the circle is osculating and the contact is second-order, yet a single contact. Lemma 5, stated for C^2 continuous ∂D, may be extended to the case where ∂D is piecewise C^2 continuous by defining the curvature of ∂D at C^2 discontinuities to be $+\infty$ at points where D is *locally convex* (points p and q in Figure 3) and $-\infty$ at points where ∂D is *locally concave* (point w in Figure 3). All the arguments of the proof of Lemma 5 follow verbatim by noting that the points of C^2 discontinuities qualify as local curvature extremum. For simplicity, in the ensuing discussion we assume that at each point of ∂D where ∂D is C^2 discontinuous, D is locally convex. The example shown in Figure 3 clearly does not satisfy this assumption. Points of type w are excluded in this work.

Our approach to packing circles consists of two phases. In the first phase, circles are

packed, one at a time, in spiraling layers inside D starting from the boundary of D. This is explained in Section 4. The second phase, which consists of a perturbation step, attempts to make room for packing more circles by simulating shaking of the container D and is explained in Section 5.

4. Initialization

The first phase of our approach consists of packing circles inside the container, one at a time, starting along ∂D. Given D and the radius r of circles, we pack the circles in D in layers, from the periphery till the center. The first layer of circles is packed along ∂D so that each circle makes either single or double tangential contact(s) with ∂D. This is explained in Section 4.1. The next layer of circles is packed so that each circle makes contact with two existing circles. This is repeated until no more circles can be accommodated. This is explained in Section 4.2. A spiral strategy of placing circles is demonstrated at [2], however, it is limited to simple container shapes such as sphere or square. Placing circles along arbitrary free-form curves requires solving algebraic constraints.

4.1. The peripheral layer

The circles in the peripheral layer are placed so that each circle makes one or more tangential contacts with ∂D. If there exists a point, $p \in \partial D$, such that $\kappa(p) > \frac{1}{r}$, then the first circle is placed in D so that it makes two tangential contacts with ∂D. The existence of such a circle stems from case (a) of Lemma 5. Figure 4(a) shows an example of such a configuration, wherein, the boundary ∂D is shown in black and the circle is shown in blue. Let $D(t)$ denote the parametrization of ∂D. Then, the center, $p = (p_x, p_y)$, of a circle with two tangential contacts with ∂D at $D(t_1)$ and $D(t_2)$ is computed by solving the following set of constraints having four variables (t_1, t_2, p_x, p_y):

$$\begin{aligned} \langle p - D(t_1), p - D(t_1) \rangle &= r^2, \\ \langle p - D(t_2), p - D(t_2) \rangle &= r^2, \\ \langle p - D(t_1), D'(t_1) \rangle &= 0, \\ \langle p - D(t_2), D'(t_2) \rangle &= 0, \\ t_1 &< t_2, \end{aligned} \tag{1}$$

where t_1 and t_2 are the contact's parameters and $\langle , \rangle$ is the standard dot-product in $\mathbb{R}^2$. Here, the first two constraints ensure that p is at a distance r from the points of contact, $D(t_i), i = 1, 2$, of the circle, with ∂D, while the third and the fourth constraints ensure that the contacts are tangential. The last inequality constraint eliminates the symmetric solutions of the type $(t_1, t_2), (t_2, t_1)$, as well as trivial solutions of the type (t_1, t_1), i.e., $t_1 = t_2$. We use $\|p - D(t)\|^2$ instead of $\|p - D(t)\|$ since the former, being algebraic, can be encoded using B-splines. We solve the above set of constraints using the multivariate solver [11] of IRIT [6].

If the curvature of ∂D is bounded from above by $\frac{1}{r}$, then by case (b) of Lemma 5, it is possible to place a circle of radius r inside (and anywhere along) D so that it makes a single tangential contact with ∂D. The center, p_0, of such a circle is computed as follows. One may sample a point in ∂D and take a normal offset into D^o at distance r, which serves as the center of the first circle. An example of such a configuration with one circle is shown in

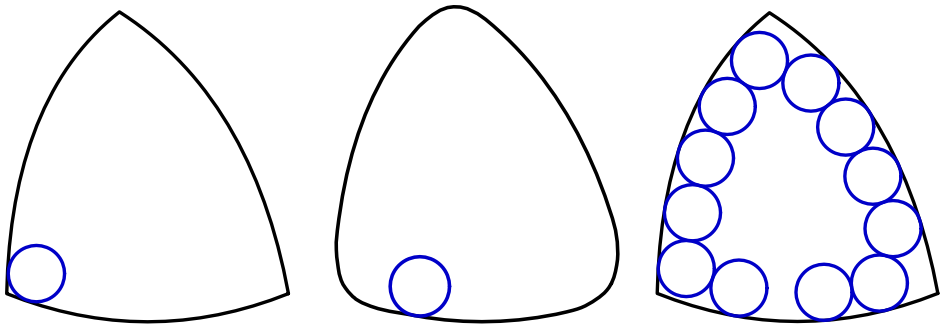


Figure 4: Packing the first layer of circles along the periphery of D. The boundary ∂D and the circle are shown in black and blue respectively.

Figure 4(b). It is readily verified that any point in ∂D may serve the purpose, since all the points of the medial axis of D are at a distance not lesser than r from ∂D.

Note that in both the above cases, there may be multiple choices for placing the first circle in D. While this choice could potentially affect the number of circles in the final result of our algorithm, empirical evidence suggests otherwise and is discussed in Section 6.

Subsequent circles are packed along ∂D so that each newly-placed circle makes two tangential contacts, either both with ∂D, or, one with ∂D and another with one of the circles in $\boldsymbol{\mathcal{C}}$. Note that if $\boldsymbol{\mathcal{C}}$ contains at least one circle, then the boundary of each component of free-space contains one or more points with curvature greater than r, namely, the points of contact between ∂D and any of the circles in $\boldsymbol{\mathcal{C}}$, as well as the points of contact between circles in $\boldsymbol{\mathcal{C}}$. At these points of contact, boundary of the free-space is C^2-discontinuous, while being locally-convex, i.e., the curvature of the boundary of free-space is $+\infty$. This is illustrated by points p, q and w in the example shown in Figure 1. While this readily follows at a point where two circles make contact (e.g. point w in Figure 1), it also follows at a point where a circle makes contact with ∂D, by the fact that the curvature of ∂D is less than that of the circle at the point of contact (e.g. points p and q in Figure 1). The existence of circles that make two tangential contacts follows by applying case (a) of Lemma 5 to the boundary of the component of the free-space of $\boldsymbol{\mathcal{C}}$ wherein the new circle is placed. This is repeated until no more circles can be placed along ∂D. Packing of peripheral layer is illustrated in Figure 4(c). It is easy to see that the centers of all the circles hence placed, lie on the offset curve of ∂D at a distance r, in the interior of D.

The center, p, of the newly introduced circle, which makes tangential contact with ∂D and with one of the circles in $\boldsymbol{\mathcal{C}}$ with center q, is obtained as the point on the circle of radius $2r$ centered at q and which is at a distance of r from ∂D (see Figure 5). Let $\delta(u)$ denote the parametrization of the circle with radius $2r$ centered at q. The point $p = \delta(u)$ is obtained as the solution of the following set of simultaneous constraints (in t and u):

$$\begin{aligned} \langle \delta(u) - D(t), \delta(u) - D(t) \rangle &= r^2, \\ \langle \delta(u) - D(t), D'(t) \rangle &= 0, \\ \langle \delta(u) - D(t), N(t) \rangle &> 0, \\ t &> t_0, \end{aligned} \tag{2}$$

where $N(t)$ is the normal to ∂D at $D(t)$ pointing into the interior of D and the point $D(t_0)$

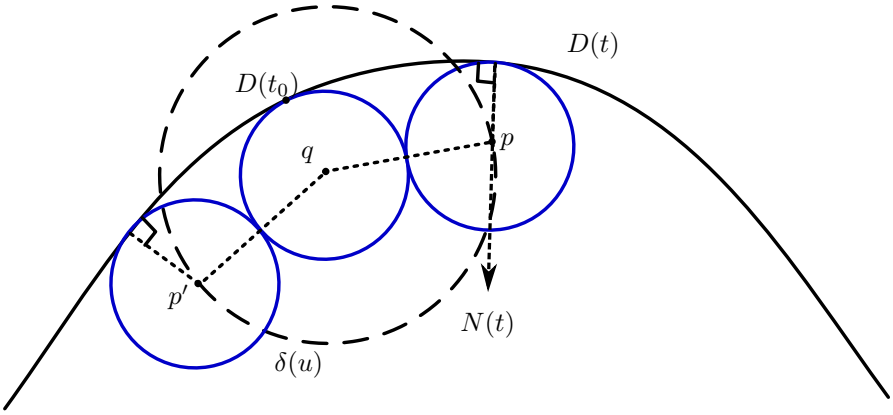


Figure 5: Placing subsequent circles along the periphery. The circle with center p is placed so that p lies on the circle of radius $2r$ centered at q and at a distance r from ∂D.

is where the circle of radius r and center q makes contact with the container. The first constraint of Equation (2) ensures that the point $\delta(u)$ is at a distance r from the point of contact $D(t) \in \partial D$ and the second constraint ensures that the contact is tangential. The third constraint, which is semi-algebraic, ensures that the point $\delta(u)$ is in the interior of D. The last constraint ensures that only one out of the two circles on either side of the circle with center q is returned, purging p' in favor of p in Figure 5. We solve the above set of constraints using the multivariate solver of IRIT [11]. The circle thus computed is validated against Criteria 1 and 2, and if passed, it is introduced into $\boldsymbol{\mathcal{C}}$.

4.2. The subsequent layers

Given a current configuration $\boldsymbol{\mathcal{C}}$ consisting of m layers of circles, the first circle in layer $m+1$ is placed so that it makes two tangential contacts with circles from layer m. The subsequent circles in layer $m+1$ are packed so that each newly packed circle makes two tangential contacts, either both with circles from layer m or one with a circle from layer m and another with a circle from layer $m+1$. Again, the existence of a circle with two contacts follows from case (a) of Lemma 5.

Let q_1 and q_2 denote the centers of two circles in $\boldsymbol{\mathcal{C}}$ with $d(q_1, q_2) \leq 4r$. The center, p_0, of the circle making two tangential contacts, one with each of the two circles with centers q_1 and q_2, is easily obtained along the bisector line of points q_1 and q_2. Denote the mid-point of q_1 and q_2 as $m_{12} = \frac{q_1+q_2}{2}$ and the distance between q_1 and q_2 as $l = d(q_1, q_2)$. Then, p_0 is obtained as one of the following two points, $p_0^i, i = 1, 2$, which satisfy Criteria 1 and 2 (see Figure 6):

$$p_0^i = m_{12} \pm \sqrt{4r^2 - \frac{l^2}{4}} \frac{\vec{n}}{\|\vec{n}\|}, \quad i = 1, 2, \tag{3}$$

where, $\vec{n}$ is $(q_1 - q_2)$ rotated counter-clockwise by $\frac{\pi}{2}$. Let *width* of a connected component of the free-space in $\boldsymbol{\mathcal{C}}$ be the radius of the maximally inscribed circle with maximal radius, within the component. Circles are packed in the above manner until no more progress is possible, i.e., the widths of all the connected components of the free-space in $\boldsymbol{\mathcal{C}}$ are less than r.

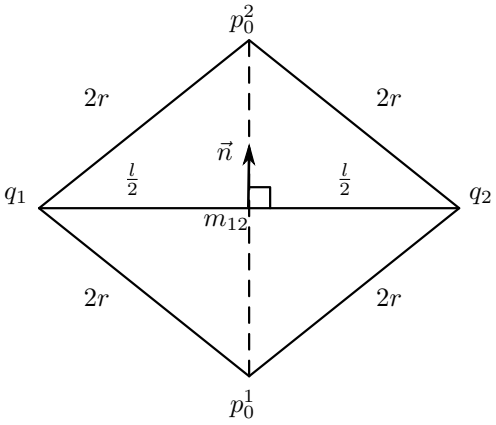


Figure 6: Placing a circle with center $p_0^i, i = 1$ or 2, which makes tangential contact with circles with centers q_1 and q_2.

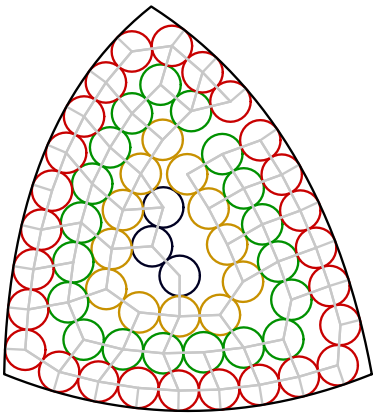


Figure 7: Initialization of packing inside a container. Circles packed in the same layer are shown in same color. Adjacency relations are indicated by gray line-segments.

Figure 7 shown an example in which a triangular container with curved edges is initialized. In this instance, there are four layers, with circles in each layer shown in red, green, yellow and blue, respectively. Adjacency relations between circles as well as between the circles and the container boundary are indicated by gray line segments.

5. The perturbation approach

Once an initial configuration $\boldsymbol{\mathcal{C}}$ is computed as described in the previous section, the next phase of our approach attempts to make room for more circles by simulating shaking of the container under gravity. A direction, θ, in 2D, is chosen at random and the circles are moved in this direction, one at a time. Each time a circle is moved, an attempt is made to insert new circles if enough room has been created in any component of the free-space in $\boldsymbol{\mathcal{C}}$. For a chosen direction θ, the perturbation phase consists of three main steps:

(i) Identify a circle $C \in \boldsymbol{\mathcal{C}}$, if any, which admits motion in direction θ. This is explained in Section 5.1.

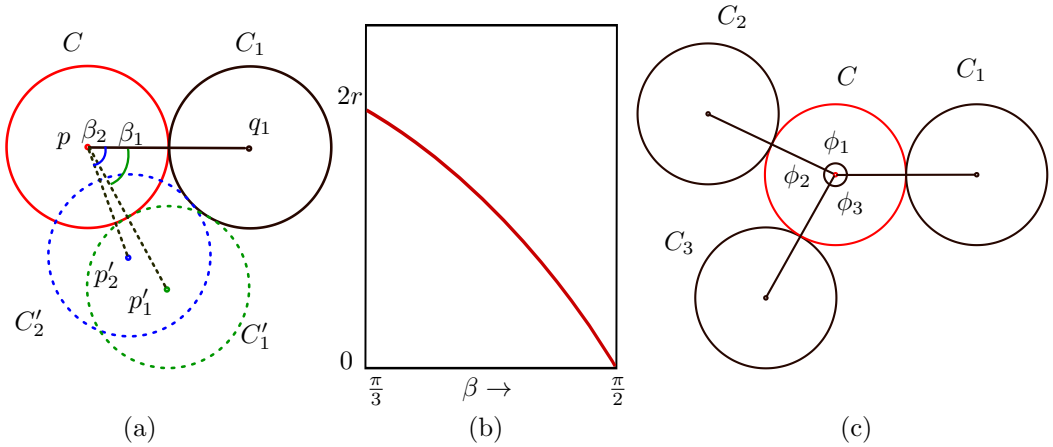


Figure 8: (a) A circle C and $C_1 \in \mathbf{N}_C$ are shown in red and black respectively. Two prospective positions after moving C are shown in green and blue, and indicated by C'_1 and C'_2, respectively. In this example, $\beta_1 = \angle_C(C_1) - \angle_C(C'_1) = \frac{\pi}{3}$ and $\|p'_1 - p\| = 2r$ while $\beta_2 = \angle_C(C_1) - \angle_C(C'_2) > \frac{\pi}{3}$ and $\|p'_1 - p\| < 2r$. (b) The lower-bound on the gouge-free distance $\|p' - p\|$ is plotted as a function of angle β. (c) A circle C shown in red with three neighbors, C_1, C_2 and C_3, shown in black. The neighbors are ordered counter-clockwise as C_1, C_2, C_3, C_1, wherein the pairs of consecutive neighbors are $(C_1, C_2), (C_2, C_3)$ and (C_3, C_1), with $\angle_C(C_1, C_2) = \phi_1, \angle_C(C_2, C_3) = \phi_2$ and $\angle_C(C_3, C_1) = \phi_3$.

(ii) Move C along θ, obtaining a new configuration $\boldsymbol{\mathcal{C}}'$, as explained in Section 5.2.

(iii) Insert new circles, if possible, in $\boldsymbol{\mathcal{C}}'$. This is explained in Section 5.3.

For each direction θ, multiple iterations of above three steps are executed, as long as any circle can be moved along θ. Let $\boldsymbol{\mathcal{C}}_i$ denote the configuration at the end of the i^{th} iteration. Execution of step (iii) in the i^{th} iteration ensures that at the beginning of step (i) in iteration $i+1$, no new circles can be added to $\boldsymbol{\mathcal{C}}_i$ without perturbing the existing circles in $\boldsymbol{\mathcal{C}}_i$.

In order to improve the efficiency of the search for the destination of a circle that is being moved, we use a few data-structures which are described next. The circles in $\boldsymbol{\mathcal{C}}$ are sorted in descending order by their projection on the line passing through origin and parallel to the direction θ. This sorted list, referred to as L_θ, determines the order in which the circles in $\boldsymbol{\mathcal{C}}$ are selected for moving along θ. Such an order ensures fewer collisions and hence, faster execution. We partition the domain D into a grid of square cells, each having a side length equal to $3r + \epsilon$ for some small $\epsilon > 0$. This choice of length is explained in Section 5.2. Each circle is inserted in all the cells of the grid with which it intersects, which can be at most four. Further, each circle, $C \in \boldsymbol{\mathcal{C}}$, maintains information about its **neighbors**, namely, other circles of $\boldsymbol{\mathcal{C}}$ which make contact with C. This list of neighbors of C is denoted by $\mathbf{N}_C$, and may contain ∂D if that is the case. Let p and q_i denote the centers of C and $C_i \in \mathbf{N}_C$, respectively. The angle subtended by q_i at p with respect to the direction $(1, 0)$ is denoted by $\angle_C(C_i)$. Note that there is a natural counter-clockwise **cyclic order** amongst the neighbors $\mathbf{N}_C$ which is imposed by the angle subtended at p, by q_i, for each $C_i \in \mathbf{N}_C$. We will call an ordered pair of neighbors, (C_i, C_j), **consecutive**, if C_j immediately follows C_i in the cyclic order. For instance, Figure 8(c) shows an example with a circle C with three neighbors, viz., C_1, C_2 and C_3 which are ordered as C_1, C_2, C_3, C_1. In this example, the ordered pair (C_1, C_2) is consecutive but (C_2, C_1) is not. We will also use objects such as **periodic intervals**, written as $[a, b]_{2\pi}$ for $a, b \in [0, 2\pi]$ and defined as $[a, b]$ if $a \leq b$ and $[b, 2\pi] \cup [0, a]$ if $a > b$.

5.1. Identifying a circle in $\boldsymbol{\mathcal{C}}$ for moving

In this section, we give a condition which must be satisfied by a circle $C \in \boldsymbol{\mathcal{C}}$ before we attempt to move C along a chosen direction θ. This is done by inspecting the angular separation between pairs of consecutive neighbors of the circle in question. This condition, coupled with the order of moving circles imposed by L_θ, results in the localization of the search space of new configurations derived from the existing configuration, thereby yielding computational efficiency. For ease of discussion, assume for the rest of this subsection that $|\mathbf{N}_C| \geq 2$, i.e., the circle C under consideration has at least two neighbors. The case when $|\mathbf{N}_C| < 2$ is explained towards the end of the subsection.

Henceforth, let p denote the center of C and let C' denote the circle obtained by moving C. The center of C' is denoted by p'. Constraining $p' - p$ to be parallel to θ would be too restrictive, resulting in little movement of circles. Instead, we only require $p' - p$ to make an acute angle with θ. Furthermore, $\|p' - p\| < 2r$, i.e., $C \cap C' \neq \emptyset$. If $C \cap C' = \emptyset$, then C' can be a whole new circle which may be inserted, violating the assumption that no such room exists. In other words, a circle may move by a distance strictly less than $2r$.

Computing C' amounts to finding contacts for C' which may be with circles from $\boldsymbol{\mathcal{C}} \setminus \{C\}$ and/or with ∂D. Note that unlike in a physics-based simulation, in our algorithm, the circles move in discrete sizeable steps. A continuous motion would lead to collisions of C with its neighbors in the intermediate positions of movement. However, in the present work, the intermediate positions are of no relevance. In order to keep the search for neighbors of C' tractable, we choose one of its neighbors from $\mathbf{N}_C$, i.e., $\mathbf{N}_C \cap \mathbf{N}_{C'} \neq \emptyset$. Such a choice does not curtail the motion of C since the motion step is repeated with C', in turn, until no further movement is possible. Let $\mathbf{N}_C \cap \mathbf{N}_{C'} = C_1$ and q_1 be the center of C_1. This is illustrated in Figure 8(a). The circles C and C_1 are shown in red and black respectively. Two prospective positions for C', viz. C'_1 and C'_2, are indicated by dotted circles in green and blue. Since $\|p' - p\|$ is bounded from above by $2r$, $\beta_i = \angle_C(C_1) - \angle_C(C'_i)$ is bounded from below by $\frac{\pi}{3}$. In the example shown in Figure 8(a), $\beta_1 = \angle_C(C_1) - \angle_C(C'_1) = \frac{\pi}{3}$ and $\|p'_1 - p\| = 2r$ while $\beta_2 = \angle_C(C_1) - \angle_C(C'_2) > \frac{\pi}{3}$ and $\|p'_1 - p\| < 2r$. The lower-bound on the gouge-free distance $\|p' - p\|$, as a function of β is plotted in Figure 8(b). Since C' must maintain contact with C_1, β is bounded from above by $\frac{\pi}{2}$.

Definition 6. For an ordered pair of consecutive neighbors (C_i, C_j) of C, the **angle of separation** of (C_i, C_j) is defined as $\angle_C(C_j) - \angle_C(C_i)$ and denoted by $\angle_C(C_i, C_j)$.

Note that $\angle_C(C_i, C_j)$ is defined only if C_i, C_j are consecutive. Further, if $\mathbf{N}_C$ has exactly two neighbors, say C_1, C_2, then $\angle_C(C_1, C_2) = 2\pi - \angle_C(C_2, C_1)$. In the example shown in Figure 8(c), $\angle_C(C_1, C_2) = \phi_1, \angle_C(C_2, C_3) = \phi_2$ and $\angle_C(C_3, C_1) = \phi_3$. The following definition gives a local, necessary condition for a circle to move along θ, in terms of angular separation of its neighbors.

Definition 7. Given a direction $\theta \in [0, 2\pi)$, we say that $C \in \boldsymbol{\mathcal{C}}$ **admits motion in direction** θ if there exists a pair of consecutive neighbors (C_i, C_j) of C such that

(i) $\angle_C(C_i, C_j) > \frac{2\pi}{3}$, and

(ii) $[\theta - \frac{\pi}{2}, \theta + \frac{\pi}{2}]_{2\pi} \cap [\angle_C(C_i) + \frac{\pi}{3}, \angle_C(C_j) - \frac{\pi}{3}]_{2\pi} \neq \emptyset$.

A circle is subjected to a sequence of moves along θ, so that in each step it makes two contacts, with other circles and/or with the container. Suppose that C is moved in the space between its neighbors C_i, C_j. Condition (i) in Definition 7 ensures that the gouge-free

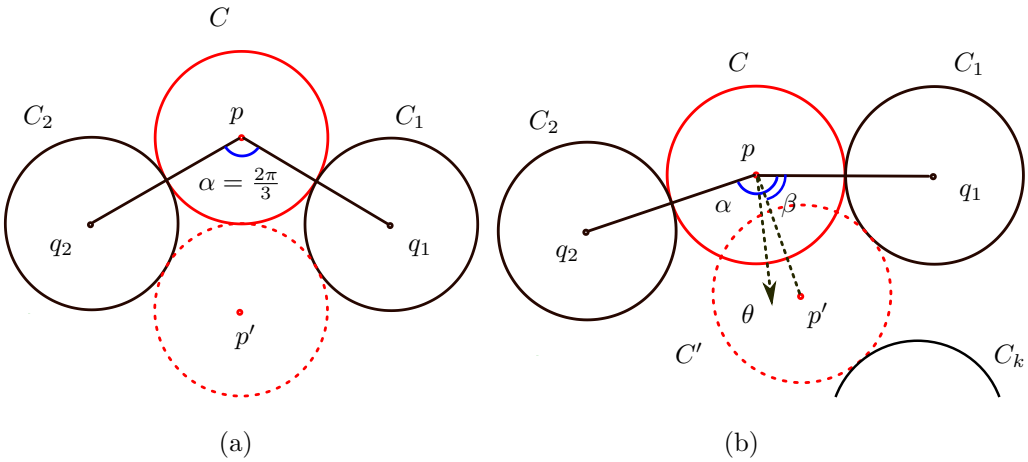


Figure 9: (a) A circle C shown in red with consecutive neighbors (C_2, C_1), shown in black, with $\angle_C(C_2, C_1)$ indicated by $\alpha = \frac{2\pi}{3}$ and $\|p - p'\| = 2r$. (b) In this case, $\alpha > \frac{2\pi}{3}$ and C admits motion along θ, shown by a dotted arrow. A plausible new position for C is indicated by dotted circle, C', shown in red with center p'. In this example, the two new neighbors of C' are C_1 and C_k. Angle $\angle_C(C_1) - \angle_C(C')$ is indicated by β.

distance $\|p - p'\| < 2r$, as explained previously and illustrated in Figure 9(a) by C shown in red, with consecutive neighbors (C_2, C_1) shown in black. In this example, $\angle_C(C_2, C_1) = \frac{2\pi}{3}$ and $\|p - p'\| = 2r$.

Condition (ii) in Definition 7 ensures that $\langle(\cos\theta, \sin\theta), p' - p\rangle > 0$, i.e., the direction $p' - p$ makes an acute angle with the intended direction of motion, θ. The angular offset of $\frac{\pi}{3}$ in the circular interval $[\angle_C(C_i) + \frac{\pi}{3}, \angle_C(C_j) - \frac{\pi}{3}]_{2\pi}$ is due to the upper-bound of $2r$ on $\|p' - p\|$, as explained previously. This is illustrated in Figure 9(b) by C shown in red and neighbors C_1, C_2 shown in black with $\angle_C(C_2, C_1)$ indicated by $\alpha > \frac{2\pi}{3}$. The new position of C is shown by a dotted circle with center p'. The direction θ is indicated by a dotted arrow. Note that, the intervals being circular, condition (ii) does not imply (i). Also, Definition 7 gives only a necessary condition for moving C and Criteria 1 and 2 must be verified for any new prospective position p'.

While it is possible to restrict p' to lie on the bisector of C_i and C_j, this would, in general, result in C' making a single contact with another circle/container. Such a configuration will result in fragmentation of free space on either side of C'. Instead, we ensure that C' makes a double contact, locally similar to the honeycomb arrangement in which a circle is adjacent to six other circles, thereby, ensuring a tight packing.

We now come to the case when $|\mathbf{N}_C| < 2$ for some $C \in \boldsymbol{\mathcal{C}}$. If $|\mathbf{N}_C| = 0$, then it is clear that C admits motion since the condition of not colliding with its neighbors is vacuously satisfied. Finally, if $|\mathbf{N}_C| = 1$, it is easy to verify that again in this case, C may always be moved in a direction that forms an acute angle with θ without colliding with its neighbor.

5.2. Moving a circle

Once a circle $C \in \boldsymbol{\mathcal{C}}$ has been identified which admits motion in the prescribed direction θ, as explained in the previous section, we find a new position, C', for C, which does not collide with any circle in $\boldsymbol{\mathcal{C}} \setminus C$. Again and for ease of discussion, we will assume that $|\mathbf{N}_C| \geq 2$. The case where $|\mathbf{N}_C| < 2$ is discussed towards the end of this subsection. C is moved in multiple steps. In the first step, a new position C' is computed so that C' makes two contacts, with other circles and/or the container. Recall that one of the neighbors is chosen from $\mathbf{N}_C$, i.e., $\mathbf{N}_C \cap \mathbf{N}_{C'} \neq \emptyset$. In particular, the common neighbor is one of the

neighbors in the pair of consecutive neighbors (C_i, C_j) of C such that $\angle_C(C_i, C_j) > \frac{2\pi}{3}$, as explained in Section 5.1. It is not difficult to show that if C admits motion along θ as per Definition 7, then it is always possible to move C to C' so that $\mathbf{N}_C \cap \mathbf{N}_{C'} \neq \emptyset$. If the new position C' again admits motion along θ, this step is repeated with C' to obtain C'' and so on. The domain D is bounded, and further, the two consecutive neighbors of C are a different pair in each iteration. Hence, such a sequence of steps is guaranteed to terminate.

It follows from case (a) of Lemma 5 that the new position C' may be computed so that it makes two tangential contacts, either both with circles in $\boldsymbol{\mathcal{C}} \setminus C$, or both with ∂D, or one with a circle from $\boldsymbol{\mathcal{C}} \setminus C$ and one with ∂D. Hence, finding the position C' is reduced to finding these two neighbors for C'. As noted already, one of the neighbors of C' is common with those of C. The task thus remains to find the second neighbor.

If the center of the first neighbor is in cell (i, j) of the grid, the search for the second neighbor needs only to be performed in the cells $\mathcal{N} := \{(i + i', j + j') | -1 \leq i' \leq 1, -1 \leq j' \leq 1\}$, i.e., the cell (i, j) and the cells which are either vertically, horizontally, or diagonally adjacent to the cell (i, j). Such a search is exhaustive since a circle in any other cell is at a distance at least $3r + \epsilon$ from the center of the first neighbor, and hence cannot serve as the second neighbor. This explains our choice of $3r + \epsilon$ as the width of the cells of the grid. Note that, while any choice of $Xr + \epsilon$, for $X \geq 3$ is feasible for the width of the cell, we choose the smallest possible value of $X = 3$, in order to keep the number of circles in each cell, small.

Consider first the case where $C_i \in \mathbf{N}_C$, which acts as the first neighbor of C', is a circle. We search each of the cells in $\mathcal{N}$ for the second neighbor of C'. To test a circle C_k in a cell in $\mathcal{N}$ as a potential second neighbor of C', we compute the center of C', p', so that C' makes tangential contact with C_i and C_k using Equation (3). The candidate C' thus computed is validated against Criteria 1 and 2. Note that C' may intersect at most four cells of the grid. Thus validation of Criterion 2 needs only to be performed locally in these four cells. There may be more than one choice for computing C' and we choose the one that is the farthest from C along θ. C' now replaces C in $\boldsymbol{\mathcal{C}}$ and has two neighbors, viz., C_i and C_k. This completes one step in moving C. C' is now inspected if it admits motion in direction θ as per Definition 7 and if so, the moving step is repeated with C'.

In order to test the boundary, ∂D, in a cell as a potential second neighbor of C', we compute the center of C' using Equation (2) so that C' makes tangential contact with C_i and ∂D. Again, the candidate thus computed is tested against Criteria 1 and 2, passing which, it replaces C.

The case where $C_i \in \mathbf{N}_C$, which acts as the first contact of C', is ∂D, is symmetric to the case where C_i is a circle. The difference being, if the second contact of C' is also ∂D. The center of C' in this case is computed using Equation (1) and validated against Criteria 1 and 2 within the respective cells, as explained before.

A single step of moving C is summarized in Algorithm 1. Algorithm 1 uses the subroutine UpdateNbrs(C', C_i, C_j) which registers C' and C_i as well as C' and C_j as neighbors of each other, as explained in the beginning of Section 5. Also, $Grid_D$ refers to the grid of square cells of width $3r + \epsilon$ for the container D. If the new circle, C', returned by Algorithm 1 admits motion, we again call Algorithm 1 with (C', θ) as the argument.

Algorithm 2 takes as input a circle C to be moved, the first neighbor C_i of the moved circle and returns the moved circle C' as well as the second neighbor of C'. It uses the subroutine CircleTangentTo(C_i, C_j) which returns the set of circles which make tangential contact to C_i as well as C_j using either Equation (1), if both of C_i, C_j are contacts with ∂D, or Equation (2), if one of C_i, C_j is a contact with ∂D and another with a circle, or

Equation (3) if both of C_i, C_j are contacts with circles.

Algorithm 1 MoveCircle(C, θ)

for all $C_i \in \mathbf{N}_C$ satisfying condition of Definition 7 **do**
  $(C', C_j) \leftarrow$ FindSecondNbr(C, C_i, θ);
  if $(C', C_j) \neq NULL$ **then**
    Remove C from $Grid_D$;
    Insert C' into $Grid_D$;
    $\boldsymbol{\mathcal{C}}' \leftarrow (\boldsymbol{\mathcal{C}} \setminus C) \cup \{C'\}$;
    UpdateNbrs(C', C_i, C_j);
    return TRUE;
  end if
end for
return FALSE

Algorithm 2 FindSecondNbr(C, C_i, θ)

$\mathcal{N} \leftarrow$ CellsContaining(C_i);
$CSet \leftarrow \emptyset$;
for all $Cell \in \mathcal{N}$ **do**
  for all $C_j \in Cell$ **do**
    $CSet \leftarrow CSet \cup \{C_j\}$;
  end for
end for
$(C_1, C_2) \leftarrow NULL$;
$MaxDist \leftarrow 0$;
for all $C_j \in CSet$ **do**
  $CTanSet \leftarrow$ CircleTangentTo(C_i, C_j);
  for all $C' \in CTanSet$ **do**
    $p \leftarrow$ Center(C);
    $p' \leftarrow$ Center(C');
    if $\langle p' - p, (\cos\theta, \sin\theta)\rangle > MaxDist$ **and** C' satisfies Criteria 1 and 2 **then**
      $(C_1, C_2) \leftarrow (C', C_j)$;
      $MaxDist \leftarrow \langle p' - p, (\cos\theta, \sin\theta)\rangle$;
    end if
  end for
end for
return (C_1, C_2);

We now come to the case where $|\mathbf{N}_C| < 2$. Recall from Section 5.1 that such circles always admit motion in any given direction θ. However, it may not always be possible to find a new position, C', so that $\mathbf{N}_{C'} \cap \mathbf{N}_C \neq \emptyset$. If that is the case, we move C in a straight line along θ till it makes contact with another circle, or the boundary. Now it is possible to continue moving C using Algorithm 1.

5.3. Inserting new circles

After a circle C has been moved through a sequence of steps as described in the previous section, we attempt to insert a new circle in D. Recall from Section 5.1 that no new circles can be packed in D before moving C. Hence, a new circle, C_n, inserted after moving C to C', must overlap with C. Let $\boldsymbol{\mathcal{C}}'$ denote the configuration obtained after moving C to C', i.e., $(\boldsymbol{\mathcal{C}} \setminus C) \cup \{C'\}$. Invoking the case (a) of Lemma 5 it follows that the position for the new circle, C_n, may be computed so that C_n makes two tangential contacts with the boundary of the free-space. If there is room to insert a new circle in the free-space, then it is always possible to insert it so that one of its contacts is with another circle in $\boldsymbol{\mathcal{C}}'$. The second contact may be either with another circle in $\boldsymbol{\mathcal{C}}'$ or with ∂D. It is readily verified that a circle $C_i \in \boldsymbol{\mathcal{C}}'$ may serve as a neighbor of C_n only if either

(i) $|\mathbf{N}_{C_i}| < 2$, or

(ii) C_i has a pair of consecutive neighbors (C_1, C_2) such that $\angle_{C_i}(C_1, C_2) \geq \frac{2\pi}{3}$.

We again employ the grid in order to narrow the search space for the two neighbors of C_n. Since C may intersect at most four cells of the grid and C_n must overlap with C, one of the neighbors of C_n may be found inside one of these four cells. The procedure for finding the second neighbor for C_n is identical to that of Algorithm 2, except that the condition $\langle p' - p, (\cos\theta, \sin\theta)\rangle > 0$ is not required.

The overall algorithm for the perturbation phase, as explained in Section 5, is summarized in Algorithm 3. The algorithm takes as input, the domain D, the configuration $\boldsymbol{\mathcal{C}}$ generated by the initialization phase as explained in Section 4 and M, the maximum number of iterations to run without any successful insertion of new circles. For each sampled direction θ, all the circles in the current configuration are moved along θ. If at least one circle is successfully moved, an attempt is made to insert new circles by the routine *AttemptInsertNewCircles* in Line 15 of Algorithm 3, as explained in Section 5.3.

Execution of one iteration of Algorithm 3 is demonstrated by an example shown in Figure 10. Figure 10(a) shows the configuration at the beginning of step 5 of Algorithm 3 and has 60 circles. Thereafter, a few circles are moved in the sampled direction θ indicated by an arrow. This intermediate configuration is shown in Figure 10(b), which again has 60 circles. The circles which are already moved are shown in green while the rest are shown in red. As can be seen, some free space is created wherein an additional circle may fit. This is done in steps 15 to 17 of Algorithm 3 and the resulting configuration, which now has 61 circles, is shown in Figure 10(c) with the new circle shown in blue.

6. Results

In this section, we compare our algorithm with some of the previous algorithms on simple symmetric containers as well as demonstrate packings in several non-convex, general containers. As is typical of any other packing algorithm, our algorithm is able, most of the time, to compute packings which are as dense as the best-known packings on one type of containers, rectangular in our case.

We randomly sample a few instances of best-known packings in rectangular containers maintained by Specht at [21]. The results of comparison for the same are shown in Table 1. The width of the rectangle in all instances is 1 while the height, which varies, is given in Table 1. The other parameters which are listed in the table are the radius of the circles, the number of circles in the best-known packing, the number of circles in the packing computed

Algorithm 3 PackPerturb(D,$\boldsymbol{\mathcal{C}}$, M)

1: $NumIter \leftarrow 0$;
2: **while** $NumIter \leq M$ **do**
3: $\theta \leftarrow$ SampleRandom$[0, 2\pi)$;
4: $L_\theta \leftarrow$ SortCircles($\boldsymbol{\mathcal{C}}, \theta$);
5: **do**
6: $Moved \leftarrow$ FALSE;
7: **for all** $C \in L_\theta$ in order **do**
8: $\mathcal{N} \leftarrow$ CellsContaining(C); //$\mathcal{N}$ is the neighborhood (grid-cells) containing C
9: **while** C admits motion along θ **do**
10: $Moved \leftarrow$ MoveCircle(C, θ);
11: **if** ! $Moved$ **then**
12: **break**;
13: **end if**
14: **end while**
15: **if** $Moved$ **and** AttemptInsertNewCircles($\mathcal{N}$) **then** //freespace may only be found in $\mathcal{N}$
16: $NumIter \leftarrow 0$;
17: **end if**
18: **end for**
19: **while** $Moved$
20: $NumIter \leftarrow NumIter + 1$;
21: **end while**

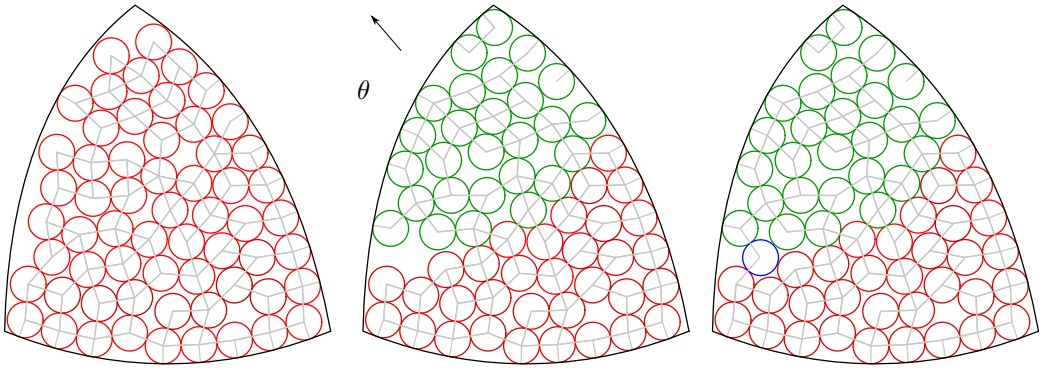
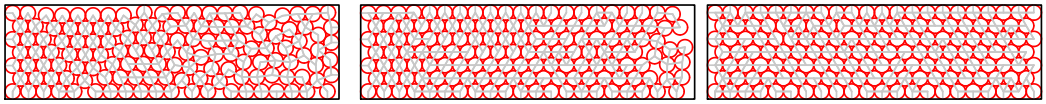


Figure 10: Inserting new circles via the perturbation step. (a) shows the configuration at the beginning of step 5 of Algorithm 3 and has 60 circles. (b) shows the configuration, again having 60 circles, after execution of steps 9 to 14 of Algorithm 3 for some circles, i.e., after moving some circles in direction θ indicated by an arrow. The moved circles are shown in green where the rest are shown in red. (c) shown the configuration with 61 circles, after the execution of steps 15 to 17 of Algorithm 3, i.e., after successfully inserting a new circle, which is shown in blue.



(a) (b) (c)

Figure 11: Configurations at the end of iterations 106, 107 and final iteration 229 for Packing Id. Rect11.

by our algorithm and the time taken by our algorithm. As can be seen in Table 1, except for two cases, namely when the number of circles in the best-known packing equals 112 and 125, our algorithm achieves the packing density as high as the best-known packing in less than an hour. Also, the iteration number in which new circles are added for each of the packings listed in Table 1, are listed in Table 2. Here, iteration number 0 refers to the initialization phase, as described in Section 4. In several instances, more than one circles are added in a single iteration, for instance, in Packing Id. Rect11, 8 circles are added in the iteration 107. Figures 11(a), 11(b) and 11(c) show the configuration at the end of iteration 106, 107 and the final iteration 229 for Packing Id. Rect11. Interestingly, our algorithm computes the packing shown in Figure 11(c) which is highly symmetric. In order to study the effect of initial packing on the final results, we computed packings for Rect8, Rect9 and Rect10 from Table 1 with two different initial packing for each. This did not change the number of circles in the final packings showing that the results are invariant of initial packings. The maximum number of iterations to execute without inserting new circles, M, which is an input parameter to Algorithm 3, is determined empirically. It is observed that the iterations of Algorithm 3 which elapse before a new circle is inserted is less than 300. Thus a value of $M = 300$ is supplied to Algorithm 3. We executed Rect8 and Rect9 (see Table 1) with a large value of $M = 20K$, yet Algorithm 3 was unable to compute packings with density as high as the best known in these two cases.

We also used other simple symmetric shapes for containers such as circles and triangles for comparison with the previous works. It turns out that our algorithm is unable to compute packings as dense as best known packings in these containers.

It is observed that for any fixed container, as the radius of packed circles goes on decreasing, the influence of the shape of the container on the packing structure reduces, especially in regions away from the container's boundary. As a consequence, the packing resembles the optimal hexagonal packing of the infinite plane. This is illustrated in Figure 12 wherein a container is packed with circles of decreasing radii.

Packings in non-convex containers of the shape of alphabets 'GMP', the continents of Australia and Africa, a hand, a horse and an elephant are shown in Figure 13. The number of circles packed and time taken for these examples are listed in Table 3. The corresponding iterations in which new circles are added are listed in Table 4. A numeric tolerance of 10^{-9} is used in all the packings reported in this paper. All the packings are computed on a single thread of a 3.4 GHz CPU. The memory footprint of all examples is under 300 MB.

7. Conclusion

This paper presents an algorithm for packing circles in general freeform containers. Our perturbation based approach captures the interaction of circles with the freeform boundary via solution of algebraic constraints. Since we are aware of no similar circle packing work in freeform containers, the effectiveness of the algorithm is demonstrated by comparing

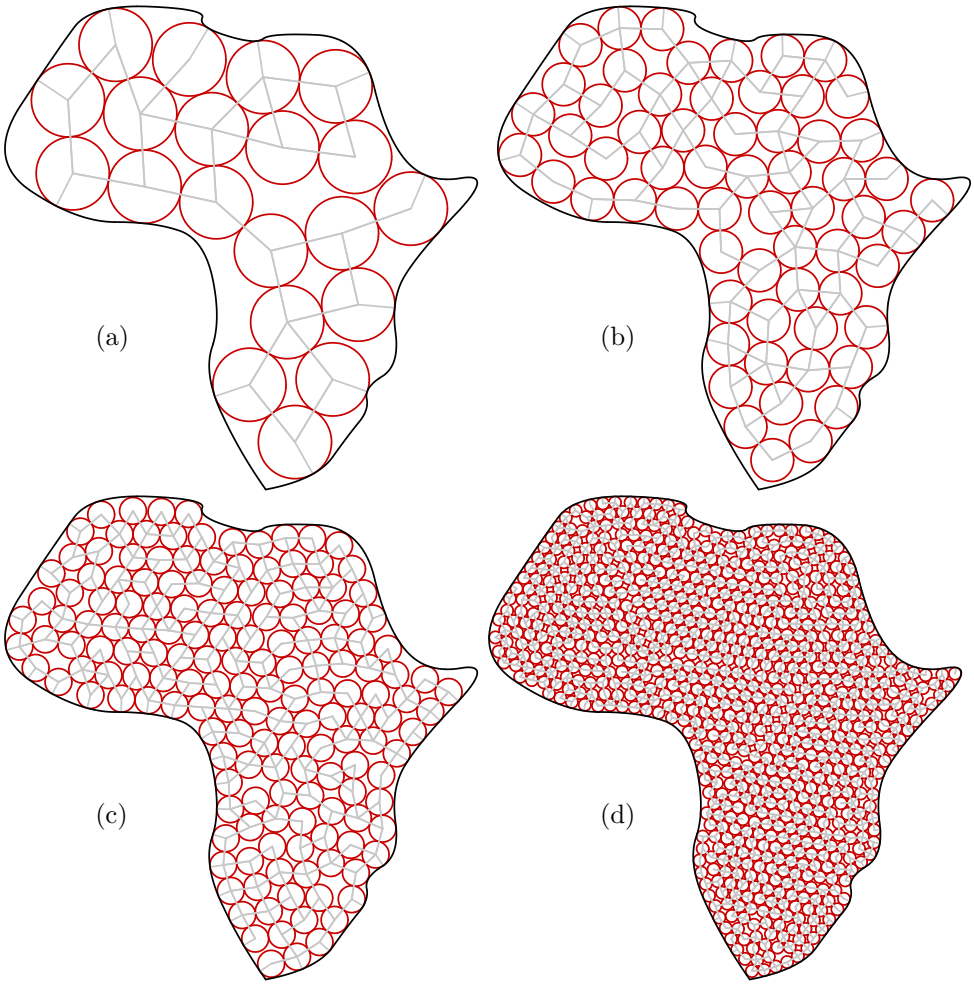


Figure 12: Packing a container with circles of decreasing radius. As the radius decreases, the packing approaches the optimal hexagonal packing of the infinite plane.

Packing Id.	Circle Radius	Rectangle Height	#CirclesBest	#CirclesOur	Time(minutes)
Rect1	0.041666666667	0.083333333333	12	12	1
Rect2	0.083333333333	0.333333333333	12	12	1
Rect3	0.125000000000	0.750000000000	12	12	1
Rect4	0.038461538462	0.143540415676	25	25	1
Rect5	0.041666666667	0.227670900631	35	35	1
Rect6	0.028571428571	0.156117189004	51	51	1
Rect7	0.023809523810	0.130097657503	62	62	1
Rect8	0.030303030303	0.375524389255	**112**	**110**	25
Rect9	0.019607843137	0.175062808437	**125**	**123**	36
Rect10	0.021739130435	0.231744652997	135	135	37
Rect11	0.022727272727	0.281643291941	151	151	45
Rect12	0.018181818182	0.193822800688	162	162	31
Rect13	0.019607843137	0.242986369518	175	175	42
Rect14	0.014705882353	0.156768441733	201	201	58

Table 1: Comparison of our packings with those of previous methods over rectangular containers from [21]. All the rectangles have unit width. Note that we fail to achieve the best known packing in the two cases shown in bold.

Pk. Id. Rect8		Pk. Id. Rect9		Pk. Id. Rect10		Pk. Id. Rect11		Pk. Id. Rect12		Pk. Id. Rect13		Pk. Id. Rect14	
Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles
0	97	0	102	0	114	0	131	0	135	0	151	0	169
1	98	1	108	4	115	14	132	1	138	2	152	1	172
12	99	31	109	5	116	21	133	2	139	14	153	2	173
45	101	33	110	7	118	24	134	4	140	18	154	18	175
46	102	36	111	8	119	70	135	11	141	40	155	20	176
47	103	37	112	9	122	105	136	33	142	45	156	24	177
48	104	40	116	13	124	106	138	35	143	51	157	31	179
59	105	42	117	17	125	107	146	37	144	54	158	32	180
62	106	50	123	18	126	109	147	40	151	57	159	35	184
68	108			23	131	165	148	42	154	59	160	36	185
100	109			44	132	229	151	48	155	60	163	39	188
113	110			51	133			54	156	66	164	40	192
				304	135			55	161	67	165	41	193
								95	162	86	166	50	194
										87	168	51	195
										90	169	101	196
										91	172	375	200
										116	175	378	201

Table 2: The iteration number in which new circles were inserted for packing in rectangular containers. Iteration number 0 refers to the initialization phase, as described in Section 4. The Packing id. refers to a particular row in Table 1. For Packing Id. 1 to 7, the listed number of circles were packed in the 0^{th} iteration, i.e. the initialization phase.

Figure Ref.	#Circles	Time(minutes)
(a)	87	46
(b)	144	57
(c)	78	32
(d)	96	63
(e)	125	77
(f)	158	81
(g)	168	94
(h)	196	112

Table 3: The running times and number of circles in packings shown in Figure 13.

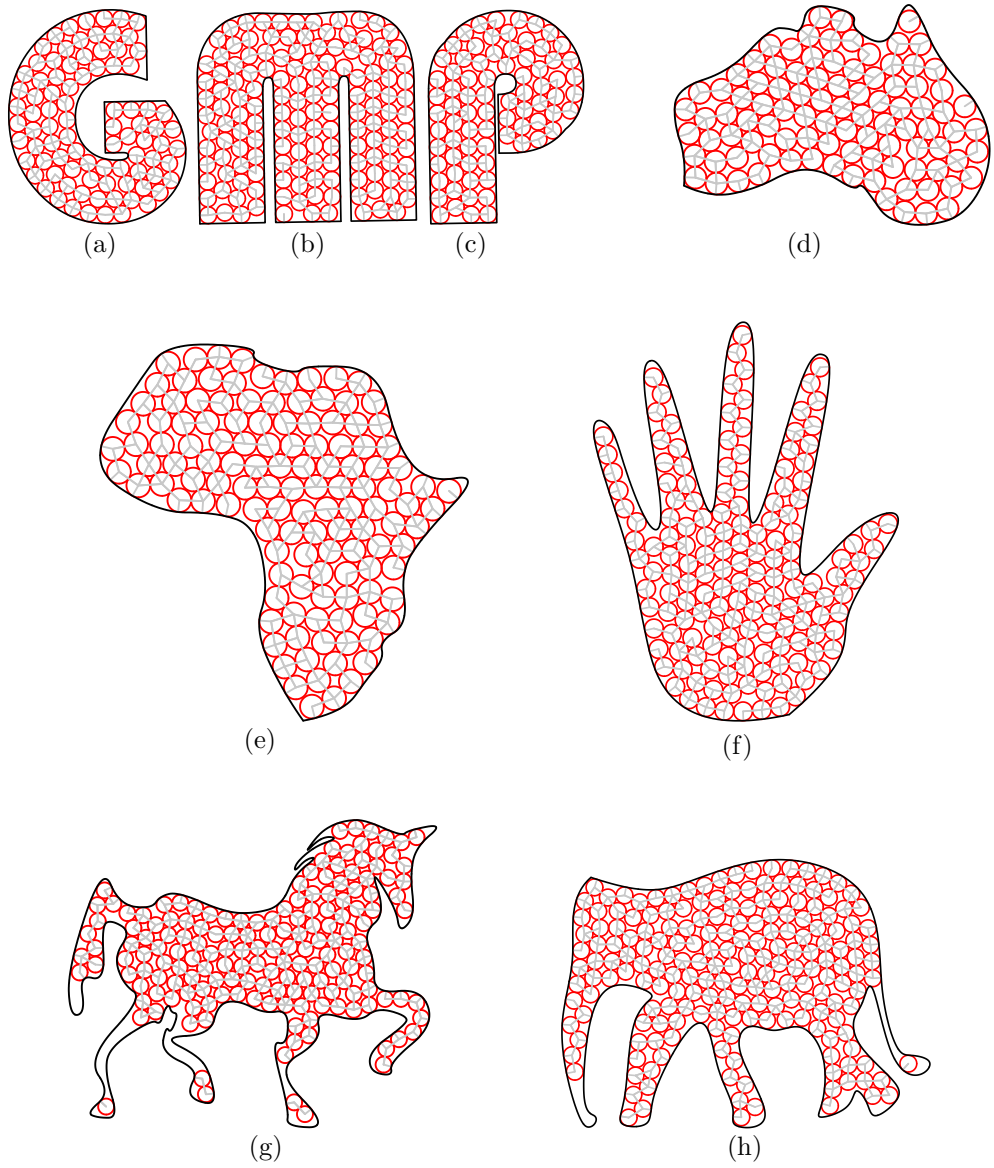


Figure 13: Packing circles inside freeform non-convex containers. (a), (b), (c) containers shaped as alphabets 'GMP'. (d) The Australian continent. (e) The African continent. (f) A hand, (g) a horse and (h) an elephant.

Figure ref. (a)		Figure ref. (b)		Figure ref. (c)		Figure ref. (d)		Figure ref. (e)		Figure ref. (f)		Figure ref. (g)		Figure ref. (h)	
Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles	Iter #	# circles
0	85	0	142	0	75	0	93	0	119	0	148	0	164	0	186
124	86	2	143	11	76	29	94	4	120	1	149	7	165	2	188
278	87	14	144	15	77	275	95	11	121	6	151	9	166	4	189
				55	78	369	96	20	122	9	154	11	167	6	191
								55	123	22	155	15	168	10	192
								242	124	24	156			22	193
								428	125	46	157			33	194
										81	158			45	195
														52	196

Table 4: The iteration number in which new circles are inserted for packings shown in Figure 13. Iteration number 0 refers to the initialization phase, as described in Section 4.

its results on simple, symmetric containers such as rectangles and triangles, with those of the previous best known packings. Our algorithm almost always achieves the best known packing on rectangular containers. The generality of the algorithm is demonstrated by packings inside freeform containers.

There are several directions along with this work may be extended. Removing the assumption of simply connectedness and local-convexity of the container will increase its range of applicability. Extending the algorithm to higher dimensional containers as well as supporting circles with variable radii could lead to interesting results and applications (e.g. stippling). Selecting boundary-specific directions of motion instead or in-addition to random directions could potentially improve the efficiency of the algorithm. Finally, the algorithm could probably be parallelized to make it more efficient.

8. Acknowledgment

This research was supported in part by the ISRAEL SCIENCE FOUNDATION (grant No.278/13). The freeform animal curves in Figure 13(g) and 13(h) were created by In-Kwon Lee and Myung Soo Kim, Postech, Korea.

References

[1] E. Birgin and F. Sobral. Minimizing the object dimensions in circle and sphere packing problems. *Computers and Operations Research*, 35(7):2357–2375, 2008.

[2] J. R. Buddenhagen. Spiral packings of circles in circles. `"http://www.buddenbooks.com/jb/pack/circle/snakes.htm"`.

[3] I. Castillo, F. J. Kampas, and J. D. Pintèrc. Solving circle packing problems by global optimization: Numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802, 2008.

[4] D. Z. Chen, X. Hu, Y. Huang, Y. Li, and J. Xu. Algorithms for congruent sphere packing and applications. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 212–221, 2001.

[5] Z. Drezner and E. Erkut. Solving the continuous p-dispersion problem using non-linear programming. *Journal of the Operational Research Society*, 46(4):516–520, 1995.

[6] G. Elber. Irit modeling environment, ver. 11. "http://www.cs.technion.ac.il/~irit/", January 2015.

[7] R. Graham. Sets of points with given minimum separation (solution to problem el921). *American Math. Monthly*, 75:192–193, 1968.

[8] R. Graham and B. Lubachevsky. Dense packings of 3k(k+1)+1 equal disks in a circle for k = 1, 2, 3, 4 and 5. In *Proc. First Int. Conf. Computing and Combinatorics, COCOON'95*, pages 303–312, 1996.

[9] R. Graham, B. Lubachevsky, K. Nurmela, and P. Ostergard. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181(1-3):139–154, 1998.

[10] A. Grosso, J. U. A. Jamali, M. Locatelli, and F. Schoen. Solving the problem of packing equal and unequal circles in a circular container. "http://www.optimization-online.org/DB_HTML/2008/06/1999.html", March 2008.

[11] I. Hanniel and G. Elber. Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations. *Computer Aided Design*, 39(5):369–378, 2007.

[12] M. Hifi and R. M'Hallah. A literature review on circle and sphere packing problems: Models and methodologies., 2009.

[13] M. Hifi, V. T. Paschos, and V. Zissimopoulos. A simulated annealing approach for the circular cutting problem. *European Journal of Operational Research*, 159(2):430–448, 2004.

[14] S. Y. Kim, R. Maciejewski, T. Isenberg, W. M. Andrews, W. Chen, M. C. Sousa, and D. S. Ebert. Stippling by example. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, pages 41–50, 2009.

[15] S. Kravitz. Packing cylinders into cylindrical containers. *Mathematics Magazine*, 40(2):65–71, 1967.

[16] C. Lopez and J. Beasley. A heuristic for the circle packing problem with a variety of containers. *European Journal of Operational Research*, 214(3):512–525, 2011.

[17] N. Mladenovic, F. Plastria, and D. Urosevic. Reformulation descent applied to circle packing problems. *Computers and Operations Research*, 32(9):2419–2434, 2005.

[18] H. Pottmann, M. H. Andreas Asperl, and A. Kilian. *Architectural Geometry*. Bentley Institute Press, 2007.

[19] H. Pottmann, M. Eigensatz, A. Vaxman, and J. Wallner. Architectural geometry. *Computers and Graphics*, 47:145–164, 2015.

[20] A. Schiftner, M. Höbinger, J. Wallner, and H. Pottmann. Packing circles and spheres on surfaces. *ACM Transactions on Graphics*, 28(5):139:1–139:8, 2009.

[21] E. Specht. Packomania web site. "http://www.packomania.com/".

[22] K. Stephenson. *Introduction to Circle Packing: The Theory of Discrete Analytic Functions*. Cambridge University Press, 2005.