Packing Curved Objects

Ignacio Salas
Mines de Nantes - LINA (UMR 6241), France
ignacio.salas@mines-nantes.fr

Gilles Chabert
Mines de Nantes - LINA (UMR 6241), France
gilles.chabert@mines-nantes.fr

Abstract

This paper deals with the problem of packing two-dimensional objects of quite arbitrary shapes including in particular curved shapes (like ellipses) and assemblies of them.

This problem arises in industry for the packaging and transport of bulky objects which are not individually packed into boxes, like car spare parts. There has been considerable work on packing curved objects but, most of the time, with specific shapes; one famous example being the circle packing problem. There is much less algorithm for the general case where different shapes can be mixed together.

A successful approach has been proposed recently in [Martinez *et al.*, 2013] and the algorithm we propose here is an extension of their work. Martinez et al. use a stochastic optimization algorithm with a fitness function that gives a violation cost and equals zero when objects are all packed. Their main idea is to define this function as a sum of $\binom{n}{2}$ elementary functions that measure the overlapping between each pair of different objects. However, these functions are ad-hoc formulas. Designing ad-hoc formulas for every possible combination of object shapes can be a very tedious task, which dramatically limits the applicability of their approach.

The aim of this paper is to generalize the approach by replacing the ad-hoc formulas with a numerical algorithm that automatically measures the overlapping between two objects. Then, we come up with a fully black-box packing algorithm that accept any kind of objects.

1 Introduction

The packing problem consists in placing n items inside a given space, such that no two items overlap. A large number of real-world applications like warehousing, logistics or parallel computing gives a great relevance to this problem. Packing has been well studied in either academic or industrial contexts, but each time considering specific shapes like bins [Lodi *et al.*, 2002], circles [Stephenson, 2005] or polytopes [Stoyan *et al.*, 2005; Egeblad *et al.*, 2009]. A lot of problems are actually studied for polytopes, some being closely related to packing, like the *collision detection* [Moore and Wilhelms, 1988; Brochu *et al.*, 2012; Mainzer and Zachmann, 2015] or the calculation of the *penetration depth* [Cameron and Culley, 1986; Dobkin *et al.*, 1993; Kim *et al.*, 2002; 2004].

Our goal is to deal with the more general situation where different shapes can be mixed, including non-convex and curved shapes.

In this context, one has to develop a generic approach for solving the packing problem. One such approach has been recently proposed in [Martinez *et al.*, 2013]. The approach splits the problem in three parts. First, they build a violation function f_{ij} for each pair (i, j) of objects. This function takes as argument the position $\vec{o} = (x, y)$ and the orientation α for both objects and gives a measure of "how much" the two objects overlap. We shall call such function an *overlapping function*.

Second, all these violations are summed up to form a global violation cost f:

$$f(\vec{o}_1, \alpha_1, \ldots, \vec{o}_n, \alpha_n) := \sum_{i>j} f_{ij}(\vec{o}_i, \alpha_i, \vec{o}_j, \alpha_j).$$

Third, the function f is minimized using a generic black-box optimization software called CMA-ES [Hansen and Ostermeier, 2001].

The experimental results they obtain are very encouraging. However, there is a serious bottleneck in their approach: the overlapping functions are ad-hoc formulas for every possible combination of object shapes. For instance, in a problem where objects are either circles, squares or triangles, one must build a formula for the 6 combinations: *circle-circle, circle-square, circle-triangle, square-square, square-triangle* and *triangle-triangle*. Some formulas are easy to obtain, like for the *circle-circle* case. Indeed, given two circles of radii r_i and r_j, the overlapping can be defined as the distance between the two centers and 0 if this distance exceeds the sum of the radii, that is:

$$f_{ij}(\vec{o}_i, \alpha_i, \vec{o}_j, \alpha_j) := \max(0, r_i + r_j - \|\vec{o}_i - \vec{o}_j\|).$$

This simple formula is rather an exception. Mixing shapes considerably complicate the task – even the *circle-rectangle* case is not so simple–. One should be convinced that, in general, these ad-hoc formulas are difficult to write. And their number grows in $\mathcal{O}(n^2)$.

We show in this paper how to replace these formulas by an algorithm. This algorithm has also been implemented and this paper reports the preliminary experimental results we have obtained on original packing problems.

The paper is organized as follows. We first formalize the problem in §2, that is, the way shapes are represented and which overlapping function we actually consider. Then, we detail in §3 and §4 an algorithm for this overlapping function. Experimental results are reported in §5, followed by concluding remarks.

2 Formalism

2.1 Object definition

We consider objects described by nonlinear inequalities. This representation choice encompasses basic geometric shapes like half-planes or ellipses. We also accept the logical and/or operators in the shape description, which means that these geometric primitives can be combined to define more complex shapes like a polygon, a half-circle or a *horseshoe* (see Figure 1). Note that non-linear inequalities give also freedom to represent a lot of fancy shapes but real-world objects are rather obtained by combining basic primitives.

With that representation, the shape of Object i can be seen as a regular constraint $c_i(\vec{p})$. This constraint means that the point $\vec{p}$ belongs to the object when the latter is placed at the origin of the frame and with a null rotation angle (see Figure 1).

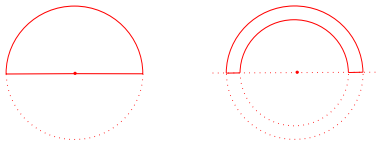

Figure 1: **Half-circle and horseshoe.** The constraint for the half-circle of radius 1 is $c(\vec{p}) \iff (\|\vec{p}\| \leq 1 \wedge y_p \geq 0)$. The constraint for the horseshoe is $c(\vec{p}) \iff (\|\vec{p}\| \leq 1 \wedge \|\vec{p}\| \geq 0.75 \wedge y_p \geq 0)$.

We will identify in the sequel a shape to a constraint and talk about "an object of shape c".

2.2 The non-overlapping constraint

An object can be translated or rotated. If Object i is placed at a position $\vec{o_i}$ and rotated by an angle α_i, we will denote by $\vec{q_i}$ and call the *parameters* of Object i the vector:

$$\vec{q_i} := (\vec{o_i}, \alpha_i) = (x_i, y_i, \alpha_i).$$

The size of $\vec{q}$ is the number of degrees of freedom. If only translation is considered, the angle is dropped and $\vec{q}$ has only two components. Otherwise, it belongs to $\mathbb{R}^3$.

Given a vector of parameters $\vec{q_i}$, the points $\vec{p}$ that belong to the translated-rotated Object i are the ones satisfying

$$c_i(R_{-\alpha_i}(\vec{p} - \vec{o_i})), \tag{1}$$

where R is the usual rotation matrix. Therefore, Objects i and j overlap iff

$$\begin{aligned} \mathsf{overlap}_{ij}(\vec{q_i}, \vec{q_j}) \iff \exists \vec{p} \in \mathbb{R}^2, \\ c_i(R_{-\alpha_i}(\vec{p} - \vec{o_i})) \wedge c_j(R_{-\alpha_j}(\vec{p} - \vec{o_j})). \end{aligned} \tag{2}$$

The non-overlapping constraint between two objects is just the negation of the latter relation.

2.3 The packing problem

The packing problem is a set of pairwise non-overlapping constraints between n objects and an additional constraint that all objects must be packed inside some *container* (like the enclosing bin in the bin packing).

One appealing aspect of our representation is that we don't actually need a specific treatment for this additional constraint. This is based on a very simple property. An object is inside the container iff this object does not overlap the complementary of the container. And the complementary of a shape is simply obtained with the negation of the underlying constraint. So, in our system, the packing problem consisting in placing n objects of shapes $c_1, \ldots, c_n$ inside a container c is reformulated as a non-overlapping constraint between $n+1$ shapes $c_1, \ldots, c_n, \neg c$.

Note that, for obvious symmetry breaking reasons, one has to fix the origin and the orientation of one object. A natural choice is then to fix that of the container.

2.4 The overlapping function

The overlapping function f_{ij} takes as input two vectors of parameters $\vec{q_i}$ and $\vec{q_j}$ and gives a measure of "how much" Objects i and j overlap. The output must fulfill two properties. First, it has to be 0 iff the two objects are disjoint. Second, it has to decrease when the two objects gets more distant.

We define the overlapping function as the distance between the current vector of parameters for Object j, and the closest vector of parameters that satisfies the non-overlapping constraint. This way, the function f_{ij} can be seen as a "distance to satisfaction" for the Object j, the parameters of Object i being considered as fixed. In fact, this function exactly matches the concept of *penetration depth* in the realm of computational geometry [Cameron and Culley, 1986; Dobkin *et al.*, 1993; Kim *et al.*, 2002; 2004].

Definition 1 (Overlapping Function)**.**

$$f_{ij}(\vec{q_i}, \vec{q_j}) = \min\{\|\vec{q_j}' - \vec{q_j}\|, \neg\, \mathsf{overlap}_{ij}(\vec{q_i}, \vec{q_j}')\}.$$

Note that the distance $\|.\|$ can be either in 2 or 3 dimensions, depending if rotation is considered or not.

As already said, the previous definition introduces a distinction between the two shapes. Object i is called the *reference* object and Object j the *moving* object. Without rotation, the roles are symmetric in the sense that:

$$f_{ij}(\vec{q_i}, \vec{q_j}) = f_{ji}(\vec{q_j}, \vec{q_i}).$$

However, the previous equality does not hold with rotation. Henceforth, we will always assume Object i (resp. j) to be the reference (resp. moving) one.

It is important to notice that our definition of the overlapping function does not coincide with the *overlapping surface*, as Figure 2 shows. Furthermore, taking the overlapping surface as a measure of overlapping would not be appropriate since the surface can reach a local minimum in a situation where the two objects still overlap (see Figure 2). In contrast, our overlapping function only reaches a minimum at 0, that

is, when the objects are disjoint. This is an important difference as this function is eventually called by a local search that can be trapped in local minima.

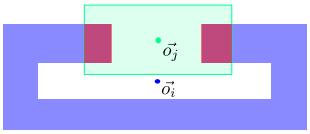

Figure 2: **Overlapping surface versus overlapping function.** The first object is the rectangle frame in blue, the second the rectangle in green. In this situation, the overlapping surface, in red, reaches a local minimum while the objects still overlap.

2.5 The overlapping region

The implementation we will propose for the overlapping function is based on the concept of *overlapping region*, a concept that was already introduced in [Salas *et al.*, 2014]. We recall this concept and then give the theoretical result that makes the connection with our definition of overlapping function.

The overlapping region simply corresponds to the set of parameters for Object j that make the objects i and j overlap, the parameters of Object i being all set to 0:

Definition 2 (Overlapping Region)**.** Given Objects i and j (resp. the reference and moving ones), the overlapping region is

$$\mathcal{S}_{ij} := \{\vec{q_j} \in \mathbb{R}^d \mid \mathsf{overlap}_{ij}(\vec{0}_{\mathbb{R}^d}, \vec{q_j})\}.$$

This concept is depicted in Figure 3.

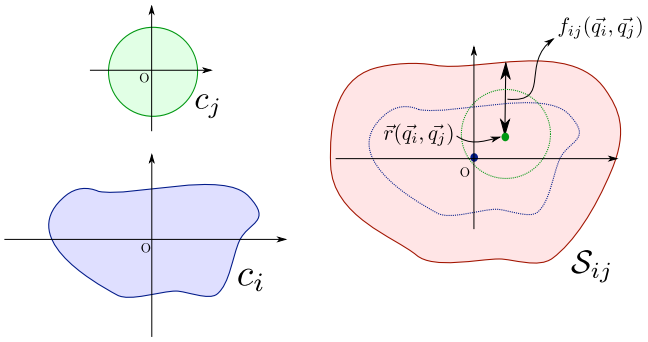

Figure 3: **Overlapping region.** For visibility, the dimension here is 2 (rotation is not considered). **(left)** Objects i (in blue) and j (in green). **(right)** The overlapping region $\mathcal{S}_{ij}$ (in red). Represents all the positions where Object j overlaps Object i, placed at the origin. The distance between the point $\vec{r}(\vec{q_i}, \vec{q_j})$ and the boundary of $\mathcal{S}_{ij}$ is the value of the overlapping function f_{ij} (Proposition 1).

We show now the relation that connects the overlapping function to the overlapping region.

Proposition 1. *Given two vectors of parameters $\vec{q_i}$ and $\vec{q_j}$, let*

$$\vec{r}(\vec{q_i}, \vec{q_j}) := (R_{-\alpha_i}(\vec{o_j} - \vec{o_i}), \alpha_j - \alpha_i). \quad (3)$$

Then

$$f(\vec{q_i}, \vec{q_j}) = \min\{\|\vec{q''} - \vec{r}(\vec{q_i}, \vec{q_j})\|,\ \vec{q''} \notin \mathcal{S}_{ij}\}.$$

Proof. By definition,

$$f(\vec{q_i}, \vec{q_j}) = \min\{\|\vec{q_j}' - \vec{q_j}\|,\ \neg\,\mathsf{overlap}_{ij}(\vec{q_i}, \vec{q_j}')\}.$$

Let us introduce the variable

$$\vec{q''} = (\vec{o''}, \alpha'') := (R_{-\alpha_i}(\vec{o_j}' - \vec{o_i}), \alpha_j' - \alpha_i).$$

We have, equivalently,

$$\vec{q_j}' = (R_{\alpha_i}\vec{o''} + \vec{o_i}, \alpha'' + \alpha_i).$$

In the one hand:

$$\begin{aligned}
&\|\vec{q_j}' - \vec{q_j}\|^2 = \|(R_{\alpha_i}\vec{o''} + \vec{o_i} - \vec{o_j}, \alpha'' + \alpha_i - \alpha_j)\|^2 \\
&= \|R_{\alpha_i}(\vec{o''} - R_{-\alpha_i}(\vec{o_j} - \vec{o_i})), \alpha'' + (\alpha_j - \alpha_i))\|^2 \\
&= \|R_{\alpha_i}(\vec{o''} - R_{-\alpha_i}(\vec{o_j} - \vec{o_i})\|^2 + |\alpha'' + (\alpha_j - \alpha_i)|^2 \\
&= \|\vec{o''} - R_{-\alpha_i}(\vec{o_j} - \vec{o_i})\|^2 + |\alpha'' + (\alpha_j - \alpha_i)|^2 \\
&\quad \text{(because rotation preserves distances)} \\
&= \|\vec{q''} - \vec{r}(\vec{q_i}, \vec{q_j})\|^2.
\end{aligned}$$

In the other hand it can be proven (see Proposition 1 in [Salas *et al.*, 2014]) that

$$\mathsf{overlap}_{ij}(\vec{q_i}, \vec{q_j}') \iff (R_{-\alpha_i}(\vec{o_j}' - \vec{o_i}), \alpha_j' - \alpha_i) \in \mathcal{S}_{ij},$$

that is,

$$\mathsf{overlap}_{ij}(\vec{q_i}, \vec{q_j}') \iff \vec{q''} \in \mathcal{S}_{ij}.$$

□

Hence, the overlapping function maps the current position of Objects i and j to the minimal distance between $\vec{r}(\vec{q_i}, \vec{q_j})$ and the boundary of the overlapping region. This is also depicted in Figure 3.

3 Main Algorithm

The algorithm we propose for the overlapping function is a direct consequence of Proposition 1. We first calculate (off-line) the overlapping region and then obtain (on-line) the value of $f(\vec{q_i}, \vec{q_j})$ from it. More precisely, the two steps are:

1. We entirely calculate the overlapping region $\mathcal{S}_{ij}$. *Calculating* this region means that an explicit representation of $\mathcal{S}_{ij}$ is computed, that is, under the form of numerical data. The data structure we use for $\mathcal{S}_{ij}$ is detailed below.
2. We calculate the minimal distance between $\vec{r}(\vec{q_i}, \vec{q_j})$ and $\mathcal{S}_{ij}$ using the previous data. The point is that this minimization problem is far more easy to solve than the original minimization problem involved in Definition 1.

3.1 Paving of the overlapping region

Let us start with the first step. The data structure we use for $\mathcal{S}_{ij}$ is called a *paving* (see Figure 4).

Definition 3 (Paving)**.** A paving of a set $\mathcal{S} \subset \mathbb{R}^d$ is a triplet $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ where $\mathcal{I}$ (for "inside"), $\mathcal{O}$ (for "outside") and $\mathcal{B}$ (for "boundary") are three sets of boxes verifying

$$\cup\mathcal{I} \subset \mathcal{S}, \quad (\cup\mathcal{O}) \cap \mathcal{S} = \emptyset \quad \text{and} \quad \cup(\mathcal{B} \cup \mathcal{I} \cup \mathcal{O}) = \mathbb{R}^d. \quad (4)$$

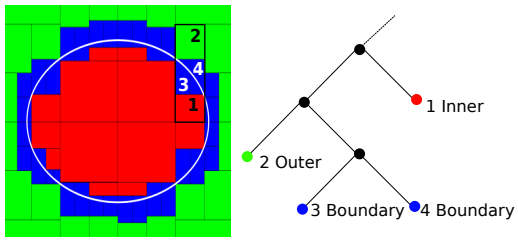

Figure 4: **Paving of an ellipse.** Boxes of $\mathcal{I}$ (resp. $\mathcal{B}$, $\mathcal{O}$) are painted in red (resp. blue, green).

To calculate this paving, we apply the strategy proposed in [Salas *et al.*, 2014]. It ensures that the total surface of the boundary (or, in other words, the precision of the paving) is less than a given value ε. Let us briefly outline this algorithm again, since we will describe an adaptation in the next section.

The algorithm is called a *paver* [Chabert and Jaulin, 2009], a recursive algorithm that starts with an arbitrarily large box $[q_j]$ and alternates three steps:

1. *(outer rejection test).* If unsatisfiability of (2) is proven for all $\vec{q_j} \in [q_j]$ ($\vec{q_i}$ being fixed), then $\mathcal{O} := \mathcal{O} \cup \{[q_j]\}$. The rejection test resorts to an embedded solver that we shall not describe further, since this part has not changed from the paper cited above.
2. *(inner inflation).* Pick randomly a point $\mathbf{q}_j \in [q_j]$ and check if it belongs to $\mathcal{S}_{ij}$. If it does, then "inflate" $\mathbf{q}_j$ to a subbox $[\mathbf{q}_j]$ proven to entirely lie inside $\mathcal{S}_{ij}$. In contrast, this part has been significantly improved and will be the topic of Section 4. Then, $\mathcal{I} := \mathcal{I} \cup \{[\mathbf{q}_j]\}$. Break the remaining part $[q_j]\backslash[\mathbf{q}_j]$ into boxes and perform a recursive call with each box.
3. if $[q_j]$ is small enough, $\mathcal{B} := \mathcal{B} \cup \{[q_j]\}$. Otherwise bisect $[q_j]$ into two sub-boxes and perform a recursive call with each boxes.

3.2 Distance to the boundary set

Once the paving $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ is calculated, we first set $\vec{r} := \vec{r}(\vec{q_i}, \vec{q_j})$ using (3) and then find the closest point to $\vec{r}$ which is not inside the overlapping region. However, because the boundary of the region is uncertain, we can actually only obtain an enclosure of this minimal distance:

$$\min_{\vec{q''} \in \cup\mathcal{B}} \|\vec{q''} - \vec{r}\| \leq f_{ij}(\vec{q_i}, \vec{q_j}) \leq \min_{\vec{q''} \in \cup\mathcal{O}} \|\vec{q''} - \vec{r}\|.$$

Clearly, the accuracy of the enclosure is related to the precision ε the paving has been generated with.

In practice, to calculate a bound, say the upper one, we consider each box $[q'']$ of the set $\mathcal{O}$ generated by the paver, and minimize the distance over that box. That is, we calculate:

$$\text{mindist}(\vec{r}, [q'']) := \min_{\vec{q''} \in [q'']} \|\vec{q''} - \vec{r}\|. \quad (5)$$

Then, we have:

$$\min_{\vec{q''} \in \cup\mathcal{O}} \|\vec{q''} - \vec{r}\| = \min_{[q''] \in \mathcal{O}} \text{mindist}(\vec{r}, [q'']). \quad (6)$$

The same holds for the lower bound.

Formula (5) is nothing different than the distance between a box and a point. This distance can be easily obtained in constant time (whatever is the size of the box), either using interval arithmetics or by a direct geometric argument.

Formula (6) can however be very time-consuming if the boxes in $\mathcal{O}$ are stored in a flat unsorted list. One has to systematically scan the whole list to get the minimal distance. This linear complexity can be easily transformed to a logarithmic complexity using a tree-structure representation, as depicted in Figure 4. The structure is then explored using standard global optimization techniques. In particular, pending nodes are stored in a heap, the criterion associated to a node being precisely its distance to $\vec{r}$. Nodes with a distance exceeding the current upper bound for the minimum can then be discarded (just like the *bounding* step in a branch & bound).

4 A new inflator for the overlapping constraint

The inflation plays a key role in the paving algorithm and thus has a strong impact on the efficiency of our packing approach.We describe in this section an original inflation technique that substantially improves packing performances.

Recall that the inflator builds, from a single vector of parameters $\mathbf{q}_j$, a box $[q_j]$ that is inside the overlapping region $\mathcal{S}_{ij}$.

In [Salas *et al.*, 2014], the inflator proposed inflates in all the dimensions of $\vec{q}$ simultaneously and turns out to be not efficient when the dimension is 3 (i.e., with rotation handled). In fact, this inflator is very sensitive to the number of times a variable appears in Equation (1), and the angle α has precisely many occurrences yield by the rotation matrix $R_{-\alpha}$. It was reported that the time required to calculate 3D pavings with this inflator was prohibitive.

Our new inflator does not suffer from the multiple symbol occurrences introduced by rotation matrices. It splits the inflation in two steps: it first inflates with respect to $\vec{o}$ and then with respect to α. We describe both steps and then show how the two inflations can be combined.

4.1 Translation

The inflation w.r.t. $\vec{o}$ is based on the following principle (see Figure 5).[1] Given initial parameters $(\mathbf{o}_j, \alpha_j)$ we first look for a vector $\mathbf{p}$ that belongs to both objects, the parameters of Objects i and j being respectively $\vec{0}$ and $\mathbf{q}_j$, that is, $\mathbf{p}$ satisfies

$$c_i(\mathbf{p}) \wedge c_j(R_{-\alpha_j}(\mathbf{p} - \mathbf{o}_j)).$$

Then, we inflate $\mathbf{p}$ inside Object i using generic inflation technique for inequalities [Chabert and Beldiceanu, 2010; Araya *et al.*, 2014]. Let $u := \mathbf{p} - \mathbf{o}_j$. The resulting box $[\vec{p}]$ can be translated to $\mathbf{o}_j$ in the sense that if o_j moves inside $[\vec{o_j}] := [\vec{p}] - \vec{u}$ then $\vec{o_j} + \vec{u}$ is both inside $[\vec{p}]$ and inside the Object j with parameters $(\vec{o_j}, \alpha_j)$.

[1] This part is similar to the old inflator applied in the 2D case.

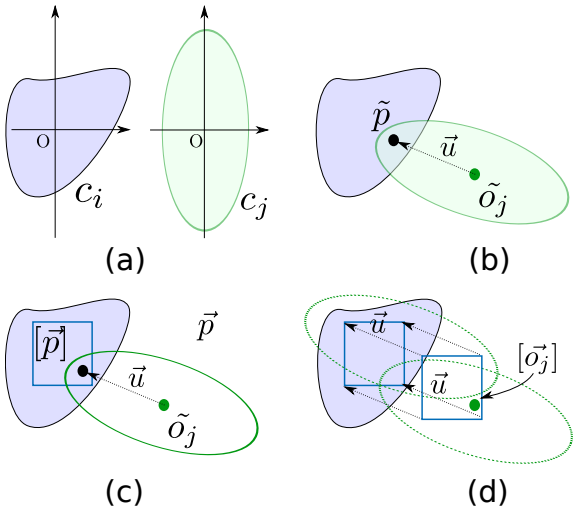

Figure 5: **Inflation w.r.t. translation.** **(a)** The shapes of Objects i (in blue) and j (in green). **(b)** A point $\tilde{p}$ at the intersection of the two objects when Object j has parameters $(\vec{o_j}, \alpha_j)$. **(c)** Inflation of $\tilde{p}$ inside Object i. **(d)** Inflation of $\vec{o_j}$.

4.2 Rotation

The inflation for the angle works as follows (see Figure 6). Given initial parameters $(\vec{o_j}, \alpha_j)$ we look for the two angles $\underline{\alpha}$ and $\overline{\alpha}$ that force vector $\tilde{p}$ (the same as before) to meet the boundary of Object j (this step is detailed in the next subsection). Then, it is clear that for every angle $\alpha_j \in [\underline{\alpha}, \overline{\alpha}]$ there is an overlapping. However, some caution is required. First, there are generally more than 2 candidate angles (see Figure 6.d), especially if Object j is non-convex. The angle values that form the narrowest interval around α_j are kept. Second, there may be no angle at all. This means that Object j can make a complete turn while containing $\tilde{p}$ (or, in other words, $\tilde{p}$ is in the inscribed circle of the object). Third, we have to take into account the 2π-periodicity. Indeed, the search space for angles is a bounded domain like $[-\pi, \pi]$ or $[0, 2\pi]$. This has the unpleasant consequence to make the inflated domain disconnected, e.g., if $\alpha_j = \frac{\pi}{2}$, $\underline{\alpha} = 0$ and $\overline{\alpha} = \frac{3\pi}{2}$ then the inflated interval $[0, \frac{3\pi}{2}]$ becomes $[-\pi, -\frac{\pi}{2}] \cup [0, \pi]$. So, in this case, we only keep the subinterval containing α_j, that is, $[-\pi, -\frac{\pi}{2}]$.

4.3 Shape Boundary

Let us explain now how the angles $\underline{\alpha}$ and $\overline{\alpha}$ are calculated. First, we have to generate a constraint c'_j for the boundary of Object j. In the simplest case of a single inequality,

$$c_j(\vec{p}) \iff f(\vec{p}) \leq 0,$$

the boundary of the object is described by the equality $f(\vec{p}) = 0$.

In the case of a conjunction of inequalities,

$$c_j(\vec{p}) \iff (f_1(\vec{p}) \leq 0 \wedge \ldots \wedge f_n(\vec{p}) \leq 0).$$

The boundary is described by a disjunction of n systems:

$$c'_j(\vec{p}) \iff (f_1(\vec{p}) = 0 \wedge f_2(\vec{p}) \leq 0 \ldots \wedge f_n(\vec{p}) \leq 0) \vee \ldots \vee (f_1(\vec{p}) \leq 0 \wedge \ldots \wedge f_{n-1}(\vec{p}) \leq 0 \wedge f_n(\vec{p}) = 0).$$

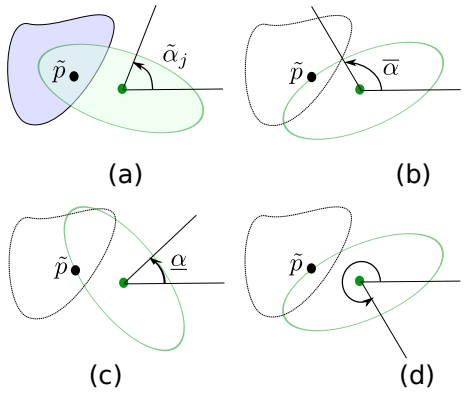

Figure 6: **Angle inflation.** **(a)** Point $\tilde{p}$ obtained with initial parameters (translation step). **(b)** The smallest angle greater than α_j that makes $\tilde{p}$ meet the boundary of object j. **(c)** The greatest angle lower than α_j with the same property. **(d)** Another angle with the same property (there are 4) that is discarded.

In the most general case of a disjunction of conjunctions of inequalities,

$$c_j(\vec{p}) \iff (f_{11}(\vec{p}) \leq 0 \wedge \ldots \wedge f_{1n_1}(\vec{p}) \leq 0) \vee \ldots \ldots \vee (f_{m1}(\vec{p}) \leq 0 \wedge \ldots \wedge f_{mn_m}(\vec{p}) \leq 0).$$

The boundary is described by a disjunction of $(n_1 + \ldots + n_m)$ conjunctions, each block of conjunctions being obtained with the previous pattern (see Figure 7).

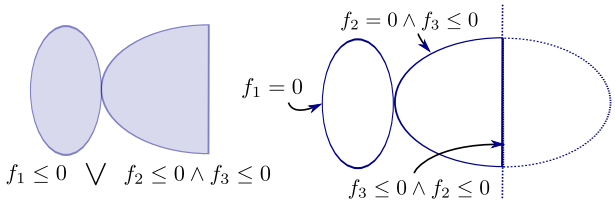

Figure 7: **Boundary of an object.**

The boundary constraint c'_j can therefore be automatically generated by a simple symbolic processing.

The angle $\overline{\alpha}$ and $\underline{\alpha}$ are then two solutions of the system $c'_j(R_{-\alpha}(\tilde{p} - \vec{o_j}))$ which is a disjunction of 1-dimensional equations (of variable α), i.e., something very easy to solve.

Note that if the shape is a disjunction with two subparts that plainly intersect (contrary to the figure), the boundary is only a subset of the generated system, in which case the angle inflation may not be optimal.

4.4 Global Inflation

We finally show that the two inflations can be combined, that is, the cartesian product of $[\vec{o_j}]$ and $[\alpha_j] = [\underline{\alpha}, \overline{\alpha}]$ is a valid inflation of c_j.

The inflation w.r.t. translation produces a box $[\vec{o_j}]$ that satisfies

$$\forall o_j \in [o_j], \quad c_i(o_j + (\tilde{p} - \vec{o_j})).$$

The inflation w.r.t. rotation produces a box $[\alpha_j]$ that satisfies

$$\forall \alpha_j \in [\alpha_j], \quad c_j(R_{-\alpha_j}(\tilde{p} - \vec{o_j})).$$

Together, they give

$$\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j], c_i(o_j + (p - \sigma_j)) \wedge c_j(R_{-\alpha_j}(p - \sigma_j)).$$

Substituting $\vec{p}$ to $(o_j + p - \sigma_j)$, we obtain:

$$\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j],\ \exists \vec{p},\ c_i(\vec{p}) \wedge c_j(R_{-\alpha_j}(\vec{p} - \vec{o_j})),$$

that is, $\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j],\ (\vec{o_j}, \alpha_j) \in \mathcal{S}_{ij}$.

5 Experimental Results

The goal of these preliminary experiments is to validate the theory and to have a first impression of how the algorithm behaves in practice. The ambition is neither to solve real-world problems nor to make an extensive performance analysis.

5.1 Setup

We have created 5 cases of increasing difficulty. In each case, n objects have to be placed with n = 10, 20 and 30. Each time, the size of the container has been adjusted manually so that the density of the resulting packing looks similar.

Case 1. This is a standard circle packing problem. The objectives are, first, to check that our algorithm also succeeds in solving this problem and, second, to measure its overhead as compared to a dedicated approach. In this circle packing instance, the radius of the i^{th} circle is equal to $\sqrt{i}$, $1 \leq i \leq n$. The radius of the enclosing circle is set to 2.1 (n = 10), 2.35 (n = 20) and 2.48 (n = 30).

Case 2. The purpose of this case is to introduce a new shape, that is, for which (to our knowledge) no dedicated solver exists. The problem is otherwise kept as simple as possible: (1) there is only one shape in addition to the enclosing shape, (2) this shape is described by a single inequality and (3) only translation is permitted (no rotation). The shape we have chosen is an ellipse of radii 1 and 0.5. The ellipses have to be packed inside a circle of radius 2.9 (n = 10), 4 (n = 20) and 5 (n = 30).

Case 3. We increase the complexity by allowing rotation in the same ellipse packing instance. With this new degree of freedom, the enclosing circle can be chosen a little bit smaller. The radius of the circle is set to 2.7 (n = 10), 3.9 (n = 20) and 4.9 (n = 30).

Case 4. We increase again the complexity by replacing the ellipse with a non-convex shape which description involves several inequalities. The shape is the horseshoe depicted in Figure 1. The enclosing shape is an ellipse of radii 2/4 (n = 10), 3.9/1.8 (n = 20) and 5/2.9 (n = 30).

Case 5. The last case is a mixed packing problem where n/2 ellipses and n/2 horseshoes have to be packed inside a circle of radius 2.9 (n = 10), 4 (n = 20) and 6 (n = 30). Again, both translation and rotation are considered.

5.2 Results

Remind that the packing process is in two steps. The *paving* step, performed off-line, and the *packing* step. Since some pavings are used in several cases, paving and packing times are reported separately. We also only give the average time for the 13×12 circle-circle pavings calculated in Case 1 and for the other pavings involving the enclosing shape (since this shape changes with n). Tables 1 and 2 summarize the paving and packing times. Figure 8 shows pictures of the results. Every paving has been calculated with a precision ε set to 0.1.

d.o.f.	*Object* i	*Object* j	*Time*
translation	circle	circle	0.01 (avg)
	ellipse	ellipse	0.16
	enclosing circle	ellipse	0.81 (avg)
translation and rotation	ellipse	ellipse	300
	enclosing circle	ellipse	2245 (avg)
	ellipse	horseshoe	1740
	horseshoe	horseshoe	3780
	enclosing circle	horseshoe	8311 (avg)
	enclosing ellipse	horseshoe	10909 (avg)

Table 1: **Paving time** (in seconds)

#objects	case 1	case 2	case 3	case 4	case 5
10	<0.1	4.5	66	103	53
20	54	32	235	372	194
30	203	53	559	1005	395

Table 2: **Packing time** (in seconds)

First, we can see that the circle packing problem has been solved in a few seconds only. However, the enclosing circle is not optimal. In [Martinez *et al.*, 2013], the circles are packed in a circle of radius 1.95 (n = 10), 2.15 (n = 20) and 2.27 (n = 30). This gap is due to the precision ε. Using this precision for paving amounts to consider that objects to pack are "enlarged by" ε. The radius we have set for the enclosing circle is actually the smallest one for which packing is successful, given the precision ε of 0.1.

Packing for the other cases has been done in the order of the minute. Above all, we see in Figure 8 that the holes introduced by non-convex objects like the horseshoe are not lost. The paving time is in the order of the hour.

6 Conclusion

We have proposed a generic algorithm for packing non convex curved objects. The approach is not limited to polyhedral shapes and does not make convex approximation. Cavities introduced by non-convexity are used to place objects.

The approach is split in two steps. The *paving* step calculates off-line the set of relative positions and orientations of one object with respect to a second one that make them overlap. The *packing* step minimizes a fitness function which is a sum of distances between points and boundaries of the previously calculated sets. These distances are fast to calculate thanks to a tree-structured representation of the sets.

The other contribution is a new inflator for the overlapping constraint, i.e., the paving step. This new inflator uses separate techniques for inflating with respect to the position and the angle, which leads to a much larger inflation (and therefore a better packing time) than with previously existing techniques.

Preliminary experimental results are encouraging. One possible future experiment could be to see how both paving

References