Electronic Supplementary Material (ESI) for Nanoscale.

Supplementary Information:

Computational Approach for Structure Generation of Anisotropic Particles (CASGAP) with Targeted Distributions of Particle Design and Orientational Order

Nitant Gupta[1], Arthi Jayaraman[1,2*]

[1] Department of Chemical and Biomolecular Engineering, 150 Academy St., University of Delaware, Newark, DE 19716. USA.

[2] Department of Materials Science and Engineering, 201 DuPont Hall, University of Delaware, Newark, DE 19716. USA.

*Corresponding author arthij@udel.edu

S1. Inverse Transform Sampling of Log-Normal Distribution for Particle Shape and Size

For log-normal distribution, the probability density function can be written as:

$$P(X) = \frac{1}{X\, s\sqrt{2\pi}} \exp\left(-\frac{(\ln(X) - m)^2}{2s^2}\right),$$

where m and s are defined by

$$m \equiv \ln\left(\frac{\mu^2}{\sqrt{\mu^2 + \sigma^2}}\right),$$

$$s^2 \equiv \ln\left(1 + \frac{\sigma^2}{\mu^2}\right).$$

The cumulative density is then obtained as:

$$F_X(x) = P(X \leq x) = \int_0^x P(X)dX = \frac{1}{2}\operatorname{erfc}\left(\frac{m - \ln(x)}{\sqrt{2}s}\right)$$

The inverse of cumulative density function is thus,

$$F_X^{-1}(u) = \exp\left(m - \sqrt{2}s\, \operatorname{erfc}^{-1}(2u)\right).$$

The random samples x_i that follow the log-normal distribution can be obtained by generating uniform random numbers u_i between 0 and 1 and setting $x_i = F_X^{-1}(u_i)$.

S2. Inverse Transform Sampling of von Mises-Fisher (vMF) Distribution for Particle Orientation

The vMF distribution for orientation $\boldsymbol{V}_d$ where $d = 3$ (for spheroids) or 4 (non-spheroidal ellipsoids), is expressed as:

$$P(\boldsymbol{V}_d) = C(\kappa)\exp(\kappa\,\boldsymbol{\Lambda}_d \cdot \boldsymbol{V}_d)$$

where $\boldsymbol{\Lambda}_d$ is the mean orientation. Here $C(\kappa)$ is a normalization constant.

As detailed in references [50-52] of the main text, random samples of $\boldsymbol{V}_d$ from the vMF distribution can be achieved in the following manner.

1. Initially assuming that the vMF distribution is around the mean direction (unit vector) $\boldsymbol{\Lambda}_d'$ which is specified as:
$$\boldsymbol{\Lambda}_d{}' = \{1,0,0,\dots\}^T.$$
2. Specify the form of $\boldsymbol{V}_d$ as:
$$\boldsymbol{V}_d = \left\{W, \qquad \boldsymbol{U}_{d-1}^T\sqrt{1-W^2}\right\}^T.$$
Here, W is a parameter in the interval $[-1,1]$ and $\boldsymbol{U}_{d-1}$ is a unit vector with $d-1$ dimensions sampled uniformly.
3. The parameter W follows a distribution $P(W)$, which is defined as
$$P(W) = c\,(1-W^2)^{\frac{d-3}{2}}\exp(\kappa W).$$
Here, c is a normalization constant.
4. Inverse transform sampling of $P(W)$ is then carried out by obtaining the cumulative density function $F_W(w)$ as
$$F_W(w) = \int_{-1}^{w} P(W)dW = \int_{-1}^{w} c\,(1-W^2)^{\frac{d-3}{2}}\exp(\kappa W)\,dW.$$
5. Step 4 can be used to also obtain c, since $F_W(1) = 1$. This indicates:
$$\frac{1}{c} = \int_{-1}^{1} (1-W^2)^{\frac{d-3}{2}}\exp(\kappa W)\,dW$$
6. Inverse transform sampling of $F_W(w)$ is then performed by generating uniform random numbers u_i between 0 and 1, and setting $w_i = F_W^{-1}(u_i)$.
7. After obtaining all w_i, $\boldsymbol{U}_{d-1,i}$ are obtained next. The vector $\boldsymbol{U}_{d-1,i}$ can be obtained by using $d-1$ standard normal variables for each i as $n_{1,i}, n_{2,i},\dots n_{d-1,i}$ and adding them to obtain $m_i = \sum_{j=1}^{d-1} n_{j,i}$. Then,
$$\boldsymbol{U}_{d-1,i} = \left\{\frac{n_{1,i}}{m_i}, \frac{n_{2,i}}{m_i}, \dots, \frac{n_{d-1,i}}{m_i}\right\}^T.$$

8. The sampled orientations $\boldsymbol{V}_{d,i}$ are obtained using the form specified in step 2. One final step is to rotate all the generated $\boldsymbol{V}_{d,i}$ to the correct mean orientation $\boldsymbol{\Lambda}_d{}' \rightarrow \boldsymbol{\Lambda}_d$.

It is noted that for $d = 3$ (spheroid orientation), the involved expressions are all analytical and straightforward to calculate. For $d = 4$, however, some numerical integration is required, which is still straightforward, since the form of $F_W(w)$ can be tabulated for a given κ value for faster calculation.

S3. Minkowski Difference of Two Polygonal Shapes

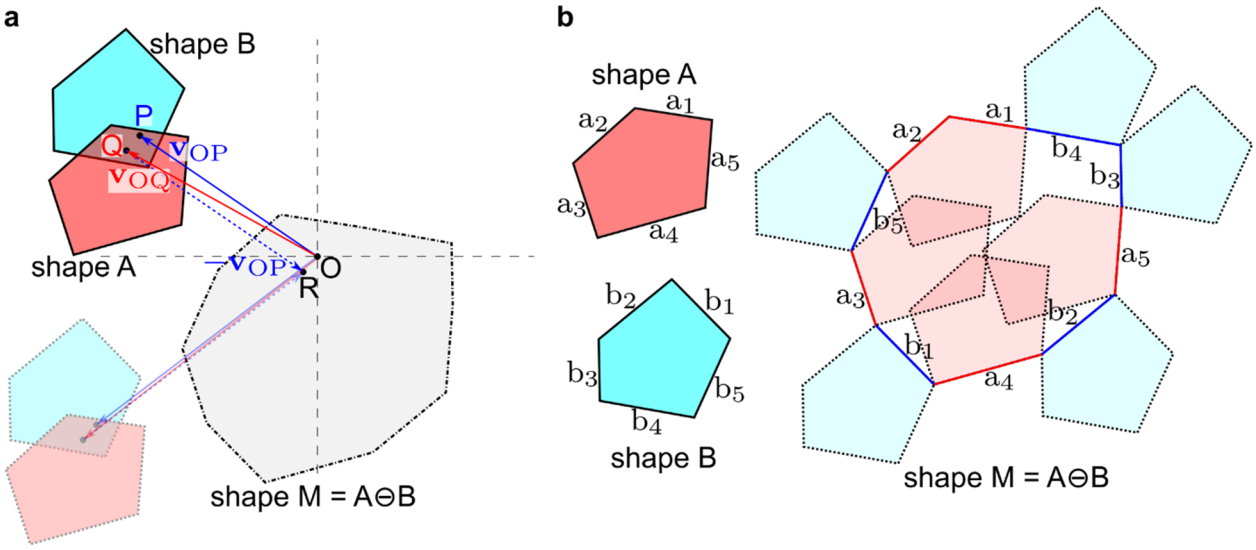

Figure S1 Minkowski difference of two shapes. *(a) The Minkowski difference M of shapes A and B, i.e., $M = A \ominus B$, is obtained by subtracting coordinates of all points in B from all points in A. Point P lies in shape B and point Q lies in shape A. Vector $\boldsymbol{v}_{OP}$ (blue vector) and $\boldsymbol{v}_{OQ}$ (red vector) are depicted to connect points P and Q to the origin O. (b) M is a composite shape of A and B consisting of all the edges in the two shapes. The edges of shape A appear in the same order (clockwise for a_1 to a_5) in M, while edges of shape B are in reverse order due to the subtraction. The shape M is invariant to relative displacements of shapes A and B, if they are not rotated.*

In **Figure S1**, a two-dimensional example is considered, where two arbitrary shaped polygons A and B are shown. If these shapes are considered as a set of points that lie inside their boundaries, then for two shapes to intersect, they must always share a common subset of these points. The Minkowski difference of these shapes, denoted as $M = A \ominus B$, is the composite shape M that is obtained when coordinates of all points in B are subtracted from coordinates of all points in A. More specifically, as shown in **Figure S1a**, when the vector v_{OP} connecting the origin O to point P (inside shape B) is subtracted from the vector v_{OQ} connecting O to point Q (in shape A), the resulting vector v_{OR} lies in proximity of the origin. In fact, if the two points P and Q approach each other such that they merge, then point R will also merge with the origin. It is easy to understand that for all possible positions of point P in shape B and point Q in shape A, the resulting point R will map the entire shape M. Given this context, when the Minkowski difference M of two intersecting shapes is computed, due to the fact that both shapes share a non-zero subset of points, the origin O must always lie inside the shape M. The objective of the GJK algorithm is to then determine whether the Minkowski difference M of two shapes contains the origin O.

S4. Demonstration of Collision Detection and Resolution Algorithms

As shown in **Figure S2**, the GJK algorithm does not compute the entirety of shape M, but rather builds a series of *simplexes* that always lie within shape M. A simplex is the simplest bounded shape in any dimension, i.e., a triangle in 3D or a tetrahedron in 4D. With each simplex built, if the origin does not lie inside it, the GJK algorithm will always search for the next simplex that may contain the origin. This process will continue until the origin is located inside the simplex, or no new simplexes can be built that will also lie inside shape M. In **Figure S2a** and **S2b** two cases are shown where the shapes A and B don't and do intersect, respectively. In case 1, the GJK algorithm is shown to build the three simplexes, after which it can't build anymore, and determines that the shapes do not intersect. Whereas, in case 2, the GJK algorithm finds the origin inside simplex 2. After an intersection is detected, the GJK algorithm stops and returns the simplex that contains the origin. Since we are also interested in resolving the intersection upon detection, we employ the use of the expanding polytope algorithm (EPA) (Ref. [39] in main text). The EPA builds upon the GJK algorithm to determine the shortest displacement vector, such that when shape B is translated by it, the intersection is resolved as shown in **Figure S2c**. The EPA thus uses the simplex containing the origin and expands it into a *polytope* that also lies within shape M, until it shares an edge (or a face in 3D), which is also the nearest to the origin. The normal vector from the origin to this edge (or face) is the required displacement vector.

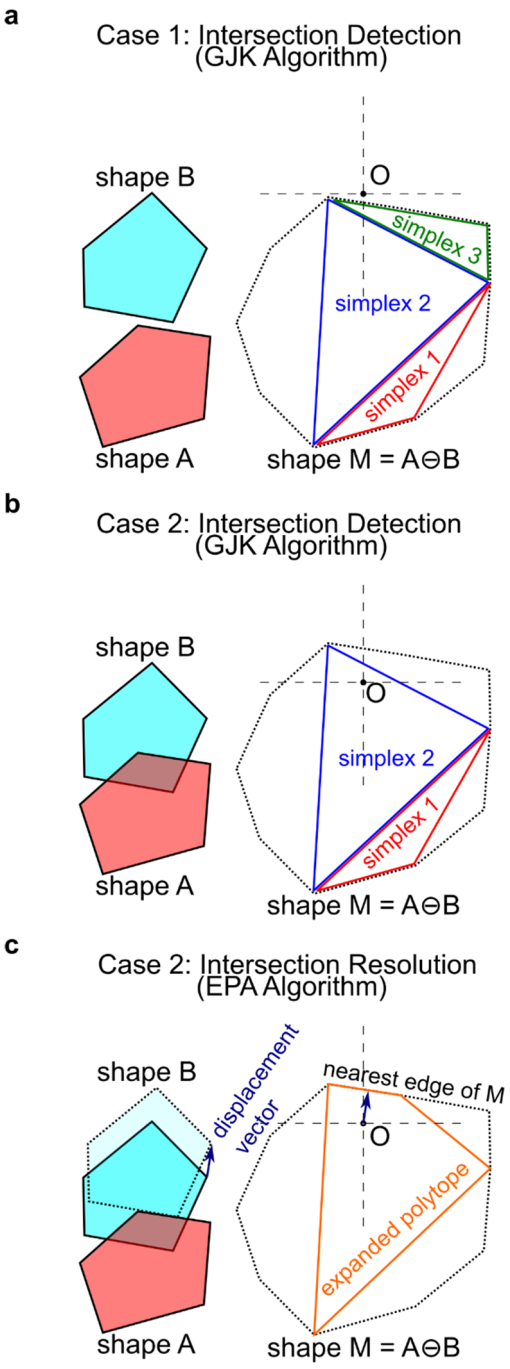

Figure S2. Demonstration of shape intersection detection and intersection resolution. *(a-b) Depicts two cases with and without shape intersection and how the GJK algorithm executes to find such intersection by constructing simplexes (triangles in 2D) until origin is located inside the simplex, or once it is detected that no further simplexes can be constructed. (c) For case 2 in (b), the EPA algorithm begins from the last simplex constructed and expands it to a polytope such that the nearest edge of M is found. The vector pointing to this edge is the displacement vector which will resolve the intersection of the two shapes.*