Computer-Aided Design 78 (2016) 14–25

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Continuous penetration depth computation for rigid models using dynamic Minkowski sums[☆]

Youngeun Lee[a], Evan Behar[b], Jyh-Ming Lien[b,a], Young J. Kim[a,*]

[a] Ewha Womans University, Seoul, South Korea
[b] George Mason University, Fairfax, VA, USA

ARTICLE INFO

Keywords:
Penetration depth
Minkowski sum
Collision detection
Convolution

ABSTRACT

We present a novel, real-time algorithm for computing the continuous penetration depth (CPD) between two interpenetrating rigid models bounded by triangle meshes. Our algorithm guarantees gradient continuity for the penetration depth (PD) results, unlike conventional penetration depth (PD) algorithms that may have directional discontinuity due to the Euclidean projection operator involved with PD computation. Moreover, unlike prior CPD algorithms, our algorithm is able to handle an orientation change in the underlying model and deal with a topologically-complicated model with holes. Given two intersecting models, we interpolate tangent planes continuously on the boundary of the Minkowski sums between the models and find the closest point on the boundary using Phong projection. Given the high complexity of computing the Minkowski sums for polygonal models in 3D, our algorithm estimates a solution subspace for CPD and dynamically constructs and updates the Minkowski sums only locally in the subspace. We implemented our algorithm on a standard PC platform and tested its performance in terms of speed and continuity using various benchmarks of complicated rigid models, and demonstrated that our algorithm can compute CPD for general polygonal models consisting of tens of thousands of triangles with a hole in a few milli-seconds while guaranteeing the continuity of PD gradient. Moreover, our algorithm can compute more optimal PD values than a state-of-the-art PD algorithm due to the dynamic Minkowski sum computation.

1. Introduction

Measuring the amount of interpenetration between overlapping models is an important problem in geometric modeling, computer graphics, computational geometry, and algorithmic robotics. A widely used distance measure for interpenetration is penetration depth (PD), which is defined as a minimum translation to separate overlapping models [1,2]. In simulated environments such as dynamics simulation, assembly planning, robot motion planning or six-degree-of-freedom haptic rendering, model overlap happens frequently due to numerical/control errors, interface latency or user-in-the-loop inherent to the environments. Such a penetration state is often considered an invalid state in computer simulation, and PD often plays a major role in recovering the invalid state to a valid, collision-free state; this type of approach is broadly classified as a penalty-based system.

It is well-known that the PD can be computed using the Minkowski sums. The Minkowski sums between $\mathcal{A}$ and $\mathcal{B}$ are defined as [3,4].

$$\mathcal{A} \oplus \mathcal{B} = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\} \tag{1}$$

$$\mathcal{A} \oplus -\mathcal{B} = \{\mathbf{a} - \mathbf{b} \mid \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\}. \tag{2}$$

If $\mathcal{A}$ and $\mathcal{B}$ overlap, their penetration depth $PD(\mathcal{A}, \mathcal{B})$ is equivalent to finding the minimum distance between the common origin of $\mathcal{A}$ and $\mathcal{B}$, $\mathbf{o}$ and the boundary surface of the Minkowski sums, $\partial(\mathcal{A} \oplus -\mathcal{B})$ [2]:

$$PD(\mathcal{A}, \mathcal{B}) = \{\min \|\mathbf{q}\| \mid \mathbf{q} \in \partial(\mathcal{A} \oplus -\mathcal{B})\}. \tag{3}$$

In other words, we can compute PD between $\mathcal{A}$ and $\mathcal{B}$ by projecting $\mathbf{o}$ onto the surface of Minkowski sums $\partial(\mathcal{A} \oplus -\mathcal{B})$.

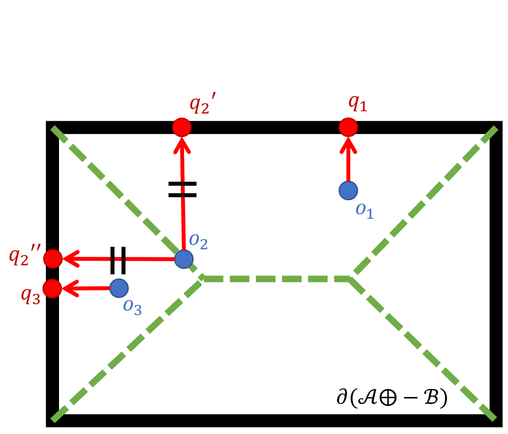
However, it is known that the definition of PD in Eq. (3) can lead to a discontinuity in the direction of PD (i.e. the gradient of PD), when $\mathbf{o}$ crosses the medial axis of $\partial(\mathcal{A} \oplus -\mathcal{B})$ [5], as illustrated

[☆] This paper has been recommended for acceptance by Scott Schaefer and Charlie C.L. Wang.

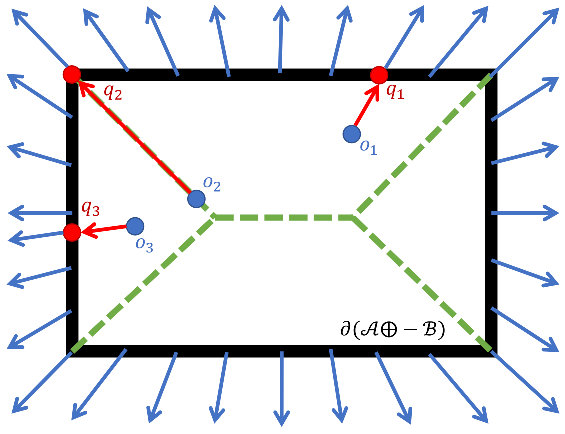
[*] Corresponding author.

E-mail addresses: youngeunlee@ewhain.net (Y. Lee), behare@gmail.com (E. Behar), jmlien@cs.gmu.edu (J.-M. Lien), kimy@ewha.ac.kr (Y.J. Kim).

http://dx.doi.org/10.1016/j.cad.2016.05.012
0010-4485/© 2016 Elsevier Ltd. All rights reserved.



(a) PD using Euclidean projection.



(b) CPD using Phong projection.

Fig. 1. In both figures, the thick rectangle is the surface of Minkowski sums and the dashed green lines represent its medial axis. The blue dots show the origin $\mathbf{o}$ as it moves from $\mathbf{o}_1$ to $\mathbf{o}_2$, and then $\mathbf{o}_3$. The red dots $\mathbf{q}_i$ show the projections of $\mathbf{o}$ on the surface of Minkowski sums, and the red arrows denote the projection directions. (a) The direction of PD using conventional Euclidean projection is not continuous when $\mathbf{o}_2$ crosses the media axis. For example, the projection suddenly jumps from $\mathbf{q}_2$ to $\mathbf{q}_2'$ even though the magnitude of PD is still continuous. (b) The surface normals (the blue arrows) on the surface of Minkowski sums are continuously defined using Phong interpolation. The projection results using Phong projection are continuous even if $\mathbf{o}_2$ is on the medial axis as the projection direction changes continuously along the corresponding surface normal. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

in Fig. 1(a). This discontinuity problem is a critical issue in any penalty-based system [6] that relies on PD as a measure of penalty response, for instance, six-degree-of-freedom haptic rendering, as it can induce serious simulation instability.

Recently, there has been a limited amount of research efforts to put into solving the discontinuity problem in PD, a new PD approach known as continuous penetration depth (CPD). Zhang et al. [7] first proposed a possible solution for CPD using spherical parameterization of configuration space. Lee and Kim [8] proposed another CPD approach using Phong projection [8]. Unfortunately both approaches are rather preliminary and limited in terms of practical applicability as they do not work when the models rotate or contain holes.

*Main results:*We present a novel algorithm to compute continuous penetration depth in real-time between two intersecting models. Our new CPD is defined using Phong projection on the subspace of Minkowski sums. This guarantees the continuous change of a PD gradient value with respect to infinitesimal rigid motion of the underlying model. In order to design an efficient CPD algorithm, we conservatively predict a solution space of CPD using a conventional PD algorithm based on Euclidean projection and construct an ϵ-ball around the predicted solution space, then compute Minkowski sums dynamically and locally inside the ϵ-ball. This dynamic Minkowski-sum method is achieved by repairing the damage on the Minkowski-sum surface due to infinitesimal rotation via the new concepts of convex map (defined in Section 4) and gap filling (Section 5). To the best of our knowledge, this is the first practical method that handles general polyhedra for Minkowski-sum computation. Then, we search for a CPD solution in the local Minkowski sums. If such a solution does not exist, we incrementally enlarge the ϵ-ball until a solution is found. We have implemented our CPD algorithm on a conventional PC platform and tested our algorithm with various benchmarking scenarios including diverse motion sequences and models of complicated geometry and topology. In our experiments, our algorithm can compute CPD for models consisting of tens of thousands triangles with holes in a few msecs while guaranteeing the continuity of PD.

*Organization:*The rest of the paper is organized as follows. We briefly survey previous work relevant to PD and dynamic Minkowski sum computation in Section 2. We discuss preliminary information and give an overview of the algorithm in Section 3. In Sections 4 and 5, we propose our algorithm to construct a local Minkowski sum, and explain how to perform Phong projection efficiently in Section 6. We show experimental results in Section 7. Finally, we conclude our work and provide a discussion on future work in Section 8.

2. Previous work

In this section, we briefly survey the work relevant to our research, namely penetration depth computation and dynamic Minkowski sum for rigid models.

2.1. Penetration depth computation for rigid models

Penetration depth. The penetration depth (PD) can be computed using a Minkowski sum-based formulation [1]. It is well-known that complexity of Minkowski sum computation is $O(n^2)$ and $O(n^6)$ for convex and non-convex models in 3D, respectively, where n is the number of facets in the objects. For convex models, exact PD can be computed using a Dobkin–Kirkpatrick hierarchy-based acceleration structure [2] and a randomized algorithm [9]. There exist various algorithms to compute approximate PD for convex models. Cameron [4] computed PD based on upper and lower bounds. Bergen [10] and Kim et al. [11] computed approximated PD in real-time.

For non-convex models, Hachenberger [12] proposed an exact PD computation algorithm based on Minkowski sums, but it is relatively slow. Various approximation algorithms have been developed for non-convex objects, but they are rather slow in practice [13,14]. Later, Je et al. [15] proposed a real-time algorithm based on iterative projection on the Minkowski sum.

Continuous penetration depth. To the best of our knowledge, there are only two works related to CPD. Zhang et al. [7] first introduced the notion of CPD and defined the CPD using a spherical parameterization of configuration space. Since this algorithm is based on sampling and machine learning, its runtime performance is very fast. However, the algorithm requires heavy preprocessing for configuration space approximation and spherical parameterization that makes it very challenging when models rotate, as the algorithm needs to recompute the entire preprocess from scratch. Moreover, the spherical parameterization does not work when the configuration space is not homeomorphic to a sphere. To circumvent these issues, Lee and Kim [8] compute CPD based on Phong projection to the surface of Minkowski sums. However, this algorithm also requires a heavy preprocessing step to compute the Minkowski sums and thus is not suited for real-world applications when the models change their orientations. Compared to these prior approaches, our new work does not require heavy preprocessing work and thus works in real-time. Moreover, the algorithm is also applicable to complicated meshes with topological holes.

Other forms of penetration depth. There exist other types of PD formulation for rigid models. For instance, the generalized PD (or simply PD_g) is defined as a minimal rigid motion to separate intersected objects [16]. PD_g can be computed using different distance metrics in configuration space such as D_g [16], S and geodesic [17], and object norm [18]. Pan et al. [19] computed PD_g using active learning. More recently, Tang and Kim [18] presented a real-time algorithm for PD_g for articulated models. An intersection volume is also a measure of model inter-penetration [20–22]. Point-wise PD [23] was also proposed using the Hausdorff distance of intersection volume.

2.2. Dynamic Minkowski sum

Dynamic Minkowski sum concerns computation involving objects under rigid transform, in particular rotation. The concept of dynamic Minkowski sum is not new. For example, there exists closely related literature on *configuration space mapping* of free-flying 2D and 3D shapes.

Given two free-flying 2D polygons P and Q, Avnaim et al. [24] proposed to compute their configuration space obstacles ($\mathcal{C}$-obst) using contact regions. A contact region is computed between a vertex of P and an edge of Q or vice versa. Their algorithm has time complexity $O(n^3m^3 \log nm)$ for polygons with n and m vertices. Similarly, Brost [25] also considered all possible contacts. Local information between the contact vertex/edge pair is used to compute the contact regions. The efficiency of these methods was later improved by focusing the computation on the discontinuity in temporal and spatial coherence where the structure of $\mathcal{C}$-obst changes [26].

At present, there are only a few published works considering rotation of 3D shapes. Donald [27] dealt with motion-planning problems with a 3D free-flying robot amongst polyhedral obstacles. Both robot and obstacles are composed of a set of convex shapes; the $\mathcal{C}$-obst are therefore a set of 6D *contact surfaces*. The intersections of these contact surfaces are computed to help the robot move along the intersection or slide from one surface to another. Mayer et al. [28] dealt with the rotation of convex polyhedra through the *criticality map*. When rotating an input polyhedron P by some angle θ about a fixed axis, the underlying structure of the Minkowski sum will change as P rotates. However, the structure does not change continuously—these structural changes only occur at a finite set of *critical values* of θ. Mayer et al. took advantage of this to compute a data structure, the criticality map, which stores these critical θ values, along with a template of the Minkowski sum structure at the critical value.

It is important to note that all of the aforementioned methods aimed to compute a complete representation of the contact space. Regardless of whether critical events are used, there is a potentially large number of values for which the structure changes, and the contact space can become quite large even for relatively simple inputs. An alternative approach is to construct the contact space and the Minkowski sum in an *on-demand* fashion. For example, Behar and Lien [29] proposed to update the facets of the Minkowski sum of convex polyhedra by identifying and correcting errors introduced through rotation. The same authors also dealt with uniform and non-uniform scaling for polygon and convex polyhedra [30]. In this paper, we propose the first known work for both convex and non-convex polyhedra.

3. Preliminaries

In this section, we provide some preliminary information related to our CPD computation including Phong projection and dynamic Minkowski sums, and also give an overview of our approach.

3.1. Phong projection

To calculate CPD, we perform Phong projection on the boundary of Minkowski sums $\partial\mathcal{M}$ where the point $\mathbf{q}$ projected from $\mathbf{o}$ to $\partial\mathcal{M}$ is collinear with the surface normal $\mathbf{n_q}$ at $\mathbf{q}$:

$$(\mathbf{q} - \mathbf{o}) \times \mathbf{n_q} = \mathbf{0}. \tag{4}$$

Euclidean projection is a kind of Phong projection. In case of Euclidean projection, points projected to the same plane have the same projection direction, which is the plane normal; however, the projection direction in Phong projection still varies for the same plane as the surface normals on the plane are continuously Phong-interpolated [31], as illustrated in Fig. 1(b). It is known that the results of Phong projection are continuous under some condition [32]. Since in our case $\partial\mathcal{M}$ is triangulated, we rely on a technique proposed by [8,32] to perform Phong projection on a set of triangles in $\partial\mathcal{M}$ to find CPD.

The Phong projection proposed by [33] defines the surface normal $\mathbf{n_q}$ in Eq. (4) by interpolating vertex normals over the planar surface. However, this normal interpolation may yield a singular normal as illustrated in Fig. 2(a). In our case, we instead interpolate tangent planes by simply rotating their basis vectors with no singular interpolation like Fig. 2(b).

A triangle $t \in \partial\mathcal{M}$ consists of three vertices $(\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)$, each of which has a tangent plane spanned by two basis vectors $\mathbf{T} \in \mathbb{R}^{2\times 3}$. A point $\mathbf{q}$ on t can be denoted as $\mathbf{q} = \lambda_1\mathbf{q}_1 + \lambda_2\mathbf{q}_2 + \lambda_3\mathbf{q}_3$ using the Barycentric coordinate $(\lambda_1, \lambda_2, \lambda_3)$. Let $\Psi(\cdot) \in \mathbb{R}^{2\times 3}$ be an interpolated tangent plane spanned by the basis $\mathbf{T}$. Then, the Phong projection from $\mathbf{o}$ to t is defined as follows:

Definition 1. A point $\mathbf{q} = (\lambda_1, \lambda_2, \lambda_3)$ on $t(\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)$ is a Phong projection of $\mathbf{o}$ on t if:

$$\Psi(\lambda_1, \lambda_2, \lambda_3)(\lambda_1\mathbf{q}_1 + \lambda_2\mathbf{q}_2 + \lambda_3\mathbf{q}_3 - \mathbf{o}) = \mathbf{0} \tag{5}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{6}$$

$$\lambda_i \geq 0. \tag{7}$$

Here, the tangent plane $\Psi(\cdot)$ should be generated continuously on a polygonal surface. To do this, we compute the per-vertex normal as an average of the surface normals of the adjacent triangles and calculate a tangent basis from the vertex normal. Then, we interpolate the tangent basis along the edges as well as over the triangle faces.

Performing Phong projection onto a triangle is equivalent to finding λ_i's satisfying Definition 1. We use the Newton's root finding method to find λ_i's satisfying Eq. (5) by initializing them to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. If there exist no λ_i's satisfying Eq. (7), we conclude that Phong projection does not exist for this triangle. In general, the existence of Phong projection is guaranteed for a point *sufficiently* close to the projected surface if the surface is a well-tessellated [32].

3.2. Dynamic Minkowski sum

Dynamic Minkowski-sum methods are usually *convolution-based* [34–37] and use the idea of *compatibility* between the primitives (vertex, edge and facet) of the inputs $\mathcal{A}$ and $\mathcal{B}$ to

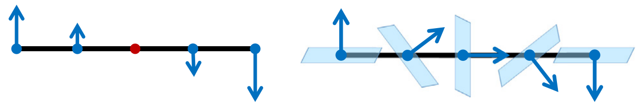


Fig. 2. Obtaining continuous surface normals. The thick line is a surface of Minkowski sums and the surface normals with an opposite direction at both ends of the surface are annotated with blue arrows. (a) Surface normals are interpolated for the rest of the surface. However, the normal at the red point becomes singular. (b) Surface normals are obtained everywhere from tangent plane (blue) interpolation with no singularity.

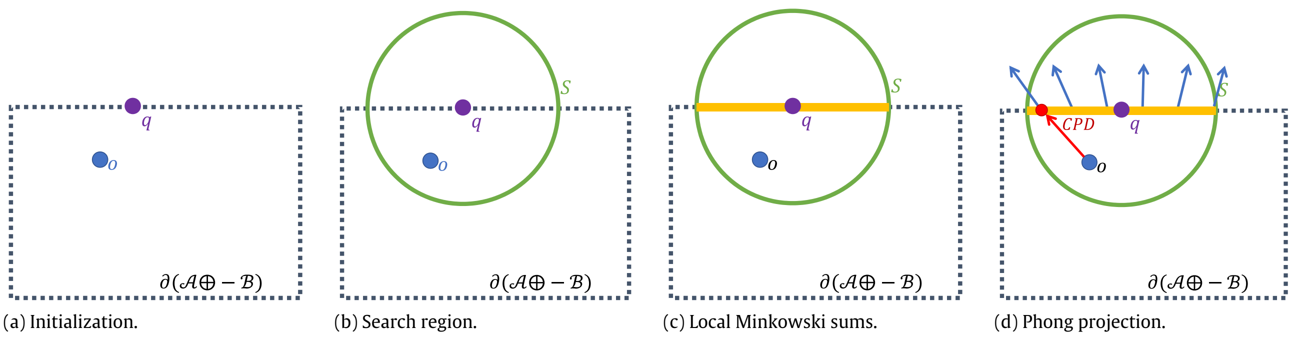

q
o
S
CPD
$\partial(\mathcal{A}\oplus-\mathcal{B})$
(a) Initialization.
(b) Search region.
(c) Local Minkowski sums.
(d) Phong projection.

Fig. 3. The dotted rectangle represents the entire surface boundary of the Minkowski sums. The blue circle is the origin **o**. (a) A candidate solution **q** (the purple circle) is obtained. (b) We define an ϵ-ball S (green circle) around **q** (c) and construct the local Minkowski sums (the yellow line) inside S. (d) We perform Phong projection within the local Minkowski sums and find the CPD (the red circle). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

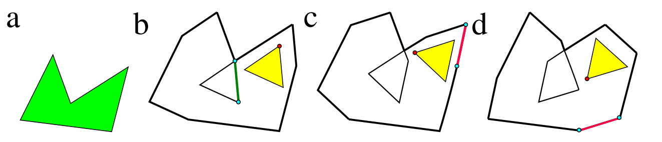


Fig. 4. The convolutions (b)–(d) between a concave shape (a) and a triangle (yellow) rotating in counter clockwise direction. These convolutions are constructed using the definition in [36]. The red dot on the triangle is the reference point. The Minkowski sums are the outer (thicker) boundary of the convolutions. The green line segment in (b) disappears after the triangle rotates from (b) to (c). This green line segment represents the error in our algorithms and should be identified and removed. The red line segments in (c) and (d) are the new segments that do not exist in (b) and (c), respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

construct the surface convolution, denoted as $\mathcal{A} \otimes \mathcal{B}$. We say that a pair of features (i.e. edge–edge or face–vertex pairs) of $\mathcal{A}$ and $\mathcal{B}$ are compatible if their Gauss maps overlap. The Gauss map of a polyhedron $\mathcal{A}$ is a function $G(\mathcal{A})$ which maps the outward normals of primitives in $\mathcal{A}$ to points, arcs and faces on the unit sphere. Consider a face $f \in \mathcal{A}$, and its outward normal $\mathbf{n}_f$. Then $G(f)$ is equal to a point $\hat{\mathbf{n}}_f$ on the unit sphere. A complete definition of Gauss map is provided in Section 4.

If $\mathcal{A}$ or $\mathcal{B}$ are non-convex, $\mathcal{A} \otimes \mathcal{B}$ is a superset of $\partial\mathcal{M} = \partial(\mathcal{A} \oplus \mathcal{B})$ and must be trimmed. For example, in [14], facets in $\mathcal{A} \otimes \mathcal{B}$ are subdivided and stitched to form continuous 2-manifold patches. Each of these patches is either on $\partial\mathcal{M}$ or entirely inside the open set of $\mathcal{M}$.

After $\mathcal{A}$ or $\mathcal{B}$ is transformed or even deformed, constructing a new Minkowski sum surface involves identifying and repairing the invalid parts of the underlying combinatorial structures. These invalid structures are called "errors" and are the results of compatible features that are no longer compatible. Further, even if a feature remains valid in $\mathcal{A} \otimes \mathcal{B}$, it might not be on $\partial\mathcal{M}$. Examples of these errors in 2D are shown in Fig. 4.

3.3. CPD algorithm overview

The goal of our CPD algorithm is to project **o** onto $\partial\mathcal{M}$ to obtain a continuous projection of **o** as in Fig. 1(b). Formally, we define our CPD as follows:

$$CPD(\mathcal{A}, \mathcal{B}) = \{\min \|\mathbf{q}\| \mid (\mathbf{q} - \mathbf{o}) \times \mathbf{n}_{\mathbf{q}} = \mathbf{0}, \forall \mathbf{q} \in \partial(\mathcal{A} \oplus -\mathcal{B})\}. \quad (8)$$

Note that it is very time-consuming to compute the surface of the Minkowski sum $\partial(\mathcal{A} \oplus -\mathcal{B})$ in its entirety due to its high combinatorial complexity. Thus, we construct Minkowski sums only locally inside a small ϵ-ball where a CPD solution may exist. A high-level description of our algorithm is given below along with an illustration (also see Fig. 3):

1. *Initialization.* Initialize a candidate solution $\mathbf{q}_0$ for CPD using a conventional PD algorithm such as [15], as CPD is close to PD for a highly tessellated surface [32]. Alternatively, if motion coherence exists in a simulated environment, one can re-use the result from the previous simulation frame as a initialization choice.
2. *Determine the search region.* We create an ϵ-ball S_i around $\mathbf{q}_i$ with a proper radius size that is likely to contain the CPD solution (Section 6.1).
3. *Construct local Minkowski sums.* We compute a subspace $\partial\mathcal{M}(S_i)$ of the Minkowski sums $\partial\mathcal{M}$ only inside S_i. If $\partial\mathcal{M}(S_i)$ has been partially computed earlier due to the nature of our iterative algorithm or simulation coherence, we only update a subset of $\partial\mathcal{M}$ newly included by S_i (Section 5).
4. *Phong projection.* We compute the tangent planes $\Psi(\cdot)$ continuously over $\partial\mathcal{M}(S_i)$ and perform Phong projection for **o** using the surface normal of $\Psi(\cdot)$ onto $\partial\mathcal{M}(S_i)$, and search for $\mathbf{q}_i$ satisfying Eq. (8) (Section 6.2).
   (a) If such $\mathbf{q}_i$ exists, a CPD solution is found.
   (b) Otherwise, we expand the search region S_i and relocate it. More specifically, we find a triangle $t \in \partial\mathcal{M}(S_i)$ and its barycenter $\mathbf{c}_t$ such that $\Psi \cdot \mathbf{c}_t$ (i.e. the left hand side of Eq. (5)) is the least for all triangles in $\partial\mathcal{M}(S_i)$. Then, we set the center of S_{i+1} to $\mathbf{c}_t$ and its radius to $\alpha\|\mathbf{c}_t\|$ for some $\alpha \in \mathbb{R}^+$.
5. *Iteration.* Steps 2–4 are repeated until we find CPD or the number of iterations reaches some maximum value.

4. The convex map

The convex map is a dual of the arrangement induced by the Gauss map. The convex map helps us identify and update the convolution when the underlying meshes rotate.

Definition 2. Let $R(A)$ be the set of regions in some arrangement A. A subdivision of the arrangement of a Gauss map G is any arrangement A for which given any region g in $R(G)$, there exists a set of regions R_g in $R(A)$ such that $g = \bigcup_{r \in R_g} r$.

Definition 3. A convex map $\mathcal{C}(\mathcal{A})$ of a non-convex polyhedron $\mathcal{A}$ is a convex polyhedron such that the arrangement of its Gauss map $G(\mathcal{C}(\mathcal{A}))$ is a subdivision of the arrangement of the Gauss map $G(\mathcal{A})$ of $\mathcal{A}$.

Although the name "convex map" is similar in name to the concept of "convex partition", the two concepts have no relationship to one another. A convex partition is a decomposition of a non-convex shape into constitution convex pieces, which can be unioned together to recover the original input shape. However, a convex map is a single convex polyhedron which is a dual of the Gauss map of the input. In general is impossible to recover the input shape correctly from the convex map alone. In order to make effective use of the convex map in correcting convolution surface errors, the convex map is annotated with references to the input polyhedron's primitives.

We will first briefly introduce the Gauss map $\mathcal{A}$ for those that are not familiar with the idea and then formally define the convex map.

4.1. Gauss map

The Gauss map of a polyhedron $\mathcal{A}$ is a function $G(\mathcal{A})$ which maps the outward normals of faces in $\mathcal{A}$ to points on the unit sphere. Consider a face $f \in \mathcal{A}$, and its outward normal $\mathbf{n}_f$. Then $G(f)$ is equal to the unit normal of f, $\hat{\mathbf{n}}_f$, which will be a point on the unit sphere. As shown in Fig. 5, edges in $\mathcal{A}$ are mapped to geodesics on the unit sphere. In particular, if $e \in \mathcal{A}$ is incident to faces $f_1, f_2 \in \mathcal{A}$, then $G(e) = (G(f_1), G(f_2))$ in the Gauss map. However geodesics are not unique. On the sphere, there are exactly two geodesics connecting every two non-antipodal points. By convention, if e is a convex edge, that is, its internal angle is less than π, then $G(e)$ is the smaller geodesic between $G(f_1)$ and $G(f_2)$. Conversely, if e is a reflex edge, having internal angle greater than π, we choose the greater geodesic (e.g. the red arc in Fig. 5(b)). It is clear that due to the nature of polyhedra that the internal angle of e cannot be exactly π, so we need not worry about the case of antipodal points, for which there are infinitely many geodesics from which to choose. Note that, these arcs do not encode directional information and therefore do not represent *tracings* [35], which always have length less than π.

Convex polyhedron. Consider the case of a manifold convex triangle mesh $\mathcal{A}$. For a given triangle $t \in \mathcal{A}$, the angles between the arcs of $G(t)$ have constraints on them.

Observation 1. *Let $\angle(a, b)$ denote the smallest angle between a and b. Given e_1, e_2 incident to the same face f, it is the case that $\angle(G(e_1), G(e_2)) = \pi - \angle(e_1, e_2)$, and consequently $\angle(G(e_1), G(e_2)) \in (0, \pi)$.*

This follows naturally from the fact that $\angle(G(e_1), G(e_2))$ is identically the angle between the normals of e_1 and e_2 when $G(e_1)$ and $G(e_2)$ have non-zero length. The following theorem falls out from these observations:

Theorem 1. *Suppose $\mathcal{A}$ is a manifold convex triangle mesh, s.t. no two adjacent faces in $\mathcal{A}$ are coplanar. Then every region in $G(\mathcal{A})$ is convex.*

Proof. Non-convex regions must contain at least one reflex vertex, and a reflex vertex by definition has an internal angle greater than π. By Observation 1, this is impossible, thus all regions in $G(t)$ must be convex. □

We can generalize this to more complicated convex meshes: suppose instead we have a manifold convex polyhedron $\mathcal{A}$ with arbitrary polygonal faces such that no two faces are coplanar. In this case, each face of $\mathcal{A}$ must be some convex polygon (since if any face of $\mathcal{A}$ is non-convex, its reflex vertices must also be incident to reflex edges, rendering $\mathcal{A}$ non-convex). Because of this, Observation 1 still holds, and so must Theorem 1. Such general convex polyhedra have Gauss maps identical to convex triangle meshes with coplanar faces.

Non-convex polyhedron. In all of the examples we have run across so far, the arrangement of the Gauss maps of non-convex polyhedra have always been identical to the arrangement of the Gauss map of some convex polyhedron, which is to say that all of the regions of the Gauss map have always been themselves convex. While we do not believe that there exist non-convex polyhedra $\mathcal{A}$ for which this does not hold, we have thus far been unable to either prove this hypothesis or construct a counter-example. Because of this, the case of non-convex regions is addressed below.

There is potentially another problem. By definition, for any convex edge e, the arclength of $G(e) < \pi$. However, a reflex edge has arclength of $G(e) > \pi$ by definition. In such a case, the arrangement would also not be identical to the Gauss map of any convex polyhedron. We call convex regions on the unit sphere for which all boundary arcs have length $< \pi$ *absolutely convex regions*.

In the case where two reflex edges in $\mathcal{A}$, r_1 and r_2, are both incident to the same face f, then $G(r_1)$ and $G(r_2)$ must intersect each other at another point aside from $G(f)$ since every pair of distinct great circles on the sphere has precisely two intersection points, separated by arcs of length π. Consequently, when two reflex edges are adjacent to each other, they can create no sub-arcs in the arrangement of $G(\mathcal{A})$ which have arc length greater than π. In this case, if any edge further intersects the sub-arc of length π, then the length of these arcs in the arrangement is not a concern.

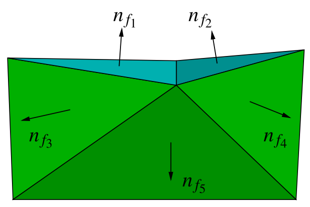
However, we may have arbitrarily many faces in $\mathcal{A}$ which have precisely one reflex edge, and it is possible to fit arbitrarily many great circle arcs with length π on the surface of a sphere without intersection.

We believe that a polyhedron exists such that there is a sub-arc of length $\geq \pi$, but have not yet found one. As a result, it remains an open problem whether there exists a non-convex polyhedron $\mathcal{A}$ such that when the *arrangement* of $G(\mathcal{A})$ is taken, there remains one or more arcs with arclength $\geq \pi$.

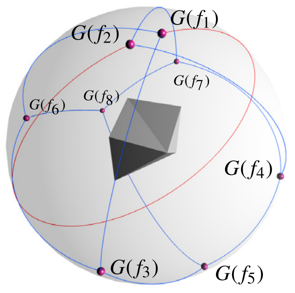
4.2. Constructing the convex map

As we will demonstrate now, we can construct a convex mapping by taking the intersection of half-spaces defined by the planes tangent to the vertices of the Gauss map. We have already proven that there exist cases for which taking the intersection of these half-spaces does not produce an identical Gauss map, but we have not yet adequately explored what changes occur when we perform this operation. We will define the convex polyhedron created by this intersection of half-spaces to be I.

Obviously, the Gauss map of I, $G(I)$ is identical to the arrangement in any absolutely convex region—the face corresponding to each tangent plane produces precisely the vertex at which it lays tangent to the sphere, and the corresponding edges all have arc length $< \pi$. However, because I must be convex in its entirety, the Gauss map must differ in two important ways:



(a) $\mathcal{A}$. The model is symmetric. On the reverse side, f_6, f_7, and f_8 correspond to f_3, f_4, and f_5 on the visible side respectively.



(b) The Gauss map of $\mathcal{A}$, $G(\mathcal{A})$. The labels denote the vertices which are Gauss maps of the respective faces from figure (a).

Fig. 5. A non-convex polyhedron $\mathcal{A}$ and its Gauss map, $G(\mathcal{A})$.

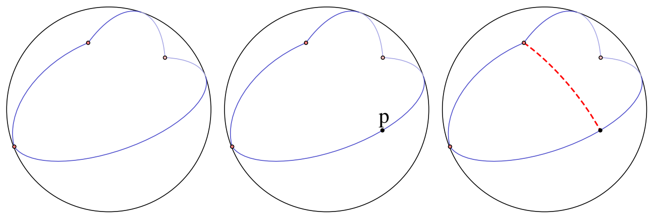


Fig. 6. Demonstration of long arc subdivision in the Gauss map (left figure). By subdividing at the midpoint, we guarantee that each sub-arc has arclength $< \pi$ (middle figure), and so taking the intersections of tangent planes only introduces phantom arcs (right figure).

1. any arcs with length $\geq \pi$ will be removed, to be replaced with an arc of length $< \pi$ if adjacency is still possible in the convex map, and
2. (*phantom arc property*) for every non-convex region R remaining in the arrangement of $G(\mathcal{A})$ after these long arcs are altered, $G(I)$ will induce arcs on the Gauss map which will cause R to become convex. We call these induced arcs *phantom arcs*.

The phantom arc property prevents any hypothetical non-convex regions from interfering with our computations. However the first property is more problematic because it can change the underlying structure of the Gauss map dramatically, in a way that would not allow us to recover the arrangement of $G(\mathcal{A})$ from it using only the union operator.

In order to preserve this structure, we subdivide these long arcs at their midpoint. Because all reflex arcs in the Gauss map have arc length $\in (\pi, 2\pi)$ by definition, the resulting sub-arcs must have arc length $\in (\frac{\pi}{2}, \pi)$. If a region adjacent to a long arc a is convex prior to subdivision, this subdivision renders the region absolutely convex, and thus it will be unchanged when the intersection of the tangent planes is taken. If a region adjacent to a is not convex, then it is subject to the phantom arc property, and will be subdivided appropriately (see Fig. 6).

Accordingly, we can now construct the convex map $\mathcal{C}(\mathcal{A})$ for any non-convex polyhedron $\mathcal{A}$ described in Algorithm 1.

Algorithm 1 Convex map algorithm

```
1: function CONVEXMAP(Polyhedron $\mathcal{A}$)
2:     Let $A$ be the arrangement of $G(\mathcal{A})$
3:     for all long arc $a \in G(\mathcal{A})$ do
4:         Insert a vertex $v$ at the midpoint of $a$.
5:     $V_c$ := an array of convex map vertices
6:     $H$ := an empty array of half-spaces
7:     for all vertex $v_i \in V_c$ do
8:         $t_i$ := the plane tangent to $a$ at $v_i \in V_c$
9:         $h_i$ := the half-space defined by $t_i$ and $v_i$
10:        Append($H, h_i$)   //This creates a set $H$ of half-spaces
   defined by tangent planes to the vertices of the arrangement.
11:    return Intersection($h_i \in H$)   //The intersection is
   guaranteed to form a convex polyhedron around the sphere.
```

5. Constructing local Minkowski sum

Given an ϵ-ball, S_i and two polyhedra, $\mathcal{A}$ and $\mathcal{B}$, we are interested in finding the local Minkowski sum boundary $\partial\mathcal{M} = \partial(\mathcal{A} \oplus -\mathcal{B}) \cap S_i$ at *interactive rates*: a task that no traditional approaches is able to accomplish. In this section, we first present a robust but slower method that determines $\partial\mathcal{M}$ without considering temporal coherence and then the idea of *gap filling* is presented to exploit temporal coherence. In our implementation, the first method is used as a bootstrap step to populate S_i and also as the last step in filling the missing facets. This new approach can also be viewed as a hybrid method that takes the advantages from the decomposition-based and convolution-based approaches.

5.1. Bootstrapping using bounding sphere hierarchy

Constructing the entire Minkowski-sum surface of two polyhedra is prohibitively slow for real time applications. Here we show that the computational efficiency can be significantly improved if the computation domain is confined within a small sphere S_i. The key idea is to cull the computation that generates the facets far from S_i. Let $\mathcal{S}(X)$ be the bounding sphere hierarchy of an object X and S_X be a node in $\mathcal{S}(X)$. The following theorem provides the main mechanism in the culling procedure.

Theorem 2. *Let $\mathcal{A}' = \mathcal{A} \cup S_{\mathcal{A}}$ be a nonempty subset enclosed by sphere $S_{\mathcal{A}}$. Similarly, $\mathcal{B}' = \mathcal{B} \cup S_{\mathcal{B}}$ is a nonempty subset in sphere $S_{\mathcal{B}}$. If $(S_{\mathcal{A}} \oplus S_{\mathcal{B}}) \cup S_i = \emptyset$, then it is guaranteed that the Minkowski sum $(\mathcal{A}' \oplus \mathcal{B}') \cup S_i = \emptyset$. Alternatively, if $(S_{\mathcal{A}} \oplus S_{\mathcal{B}}) \subset S_i$, then it is guaranteed that the Minkowski sum $(\mathcal{A}' \oplus \mathcal{B}') \subset S_i$.*

Note that $S_{\mathcal{A}} \oplus S_{\mathcal{B}}$ is simply a sphere with radius $r_{\mathcal{A}} + r_{\mathcal{B}}$ centered at $\mathbf{c}_{\mathcal{A}} + \mathbf{c}_{\mathcal{B}}$, where r_X and $\mathbf{c}_X$ are the radius and the center of the sphere X. Theorem 2 implies a culling procedure that starts from a pair of root nodes of $\mathcal{S}(\mathcal{A})$ and $\mathcal{S}(\mathcal{B})$. If $(S_{\mathcal{A}} \oplus S_{\mathcal{B}})$ is neither entirely inside nor outside S_i then all children pairs of $S_{\mathcal{A}}$ and $S_{\mathcal{B}}$ are further tested.

The culling procedure can be further optimized near the bottom of the hierarchies. If $S_{\mathcal{A}}$ encloses a single triangle $t_{\mathcal{A}}$ of $\mathcal{A}$, and $S_{\mathcal{B}}$ encloses only $t_{\mathcal{B}}$, then it is not difficult to see that the smallest bounding sphere of $t_{\mathcal{A}} \oplus t_{\mathcal{B}}$ can be much smaller than $S_{\mathcal{A}} \oplus S_{\mathcal{B}}$, for example, when $t_{\mathcal{A}}$ and $t_{\mathcal{B}}$ are long sliver triangles perpendicular to each other. Therefore, when the culling process reaches the bottoms of $\mathcal{S}(\mathcal{A})$ and $\mathcal{S}(\mathcal{B})$, a sphere tightly enclosing $t_{\mathcal{A}} \oplus t_{\mathcal{B}}$ is constructed and compared to S_i.

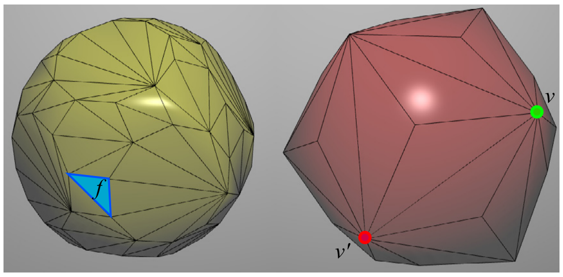


Fig. 7. Convex maps are used to identify and repair convolution errors. The (blue) face f on the left convex map was compatible with the vertex v on the right before rotation. After rotation, our method finds f's new associated vertex v' using gradient decent on the convolution of the convex maps (see details in Appendix A). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

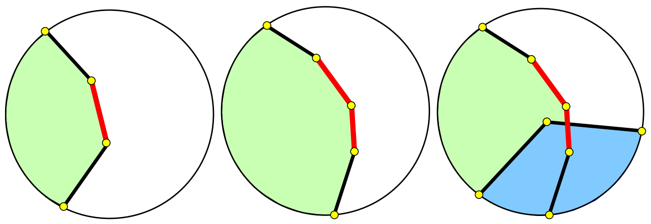


Fig. 8. A red edge of the convolution (left) is identified as an error and repaired (middle). The ϵ-ball may contain facets that are not continuously connected to the existing facets. For example the region illustrated in blue (right). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Given a pair of internal nodes $S_{\mathcal{A}}$ and $S_{\mathcal{B}}$, their enclosed subsets $\mathcal{A}'$ and $\mathcal{B}'$ contain multiple facets and can be convoluted using their convex maps $C(\mathcal{A})$ and $C(\mathcal{B})$. We then gather the local convolution facets inside S_i, and these facets are subdivided and stitched into 2-manifold patches in a way such that each patch can be classified as either on the Minkowski-sum surface or in its interior using the method proposed in [14].

5.2. Gap filling and error repair

Let $\partial\mathcal{M}_{i-1}$ denote the local Minkowski sum constructed in iteration $i - 1$. As we pointed out in Section 3.2, the combinatorial structures of $\partial\mathcal{M}_{i-1}$ may become invalid after $\mathcal{B}$ rotates and result in convolution errors and Minkowski-sum errors. In this section, we will present a method that constructs $\partial\mathcal{M}_i$ for the iteration i by repairing these errors. Note that we do not make any assumptions regarding the amount of rotation applied to $\mathcal{B}$. In some sense, the computation time of our approach is sensitive to the amount to rotation. Larger amount of rotation between consecutive frames leads to bigger errors, and thus longer computation.

Repairing convolution errors. The convolution errors can be identified using convex map. Removing convolution errors (i.e. incompatible features) results in gaps in the convolution surface. These gaps need to be filled with new compatible features. Finding these features using convex map is similar to that of constructing the dynamic Minkowski sum of convex polyhedra, therefore we leave the details to Appendix A. Fig. 7 illustrates an example of how convex maps are used.

The repaired convolution is guaranteed to form 2-manifold patches inside S_i. However, because the repairs start from the remaining valid convolution facets, it is possible that the recovered facets are incomplete. Fig. 8 illustrates such an example. As we will discuss next, these missing facets may be discovered when we repair the Minkowski-sum errors.

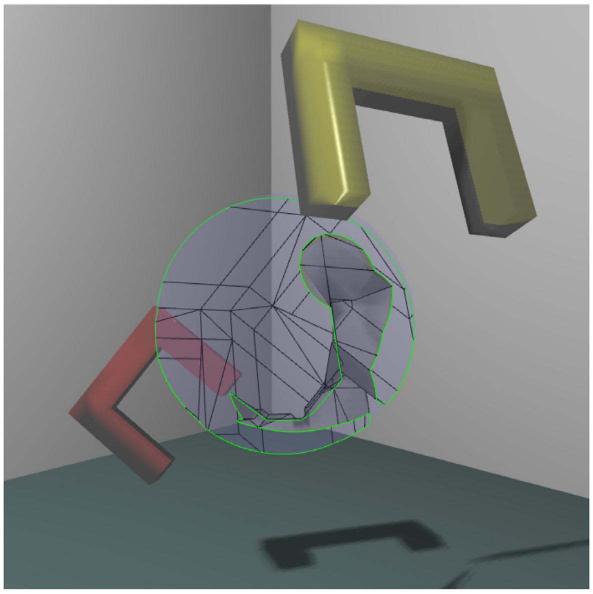


Fig. 9. Minkowski sum of two U shapes. The boundaries of the Minkowski sum (colored in green) must be on the surface of S_i. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Repairing Minkowski-sum errors. Given the repaired convolution surface, the surface again can be subdivided into 2-manifold patches and each patch can be classified as either on the Minkowski-sum surface or in the interior.

Observation 2. *A convolution patch classified as Minkowski-sum surface must form a 2-manifold whose boundaries must be on the surface of S_i. An example is shown in Fig. 9.*

If this property is violated, that means some convolution facets are missing and are not continuously connected to the existing convolution facets inside the ball (i.e. the example shown in Fig. 8). To identify these missing convolution facets, we let X be a 2-manifold convolution patch. Assume that there exists at least one point $x \in \partial X \setminus \partial S_i$. We construct a query ball $S(x)$ centered at x that is furthest away from ∂S_i and has radius equal to the distance between x and ∂S_i. Finally, the procedure described in Section 5.1 is invoked to construct the surfaces in $S(x)$.

6. Phong projection on Minkowski sums

In the previous section, we explained how to dynamically construct Minkowski sums, and now we explain how we efficiently perform Phong projection on the surface of the Minkowski sums.

6.1. Setting up the search region

As described in the algorithmic overview in Section 3.3, we compute Minkowski sums $\mathcal{M}(S)$ only within a small ϵ-ball S, as computing the entire Minkowski sum is very costly. S is defined by its center $\mathbf{c} \in \mathbb{R}^3$ and radius $r \in \mathbb{R}^+$. Ideally, we need $\mathbf{c}$ close to the solution of CPD that would make r smallest, and require minimal construction of the Minkowski sums.

Our algorithm starts with an initial solution $\mathbf{q}_0$ and repeatedly iterates over solution candidates $\mathbf{q}_i$'s. We can safely assume that the actual CPD solution is close to these solution candidates, since the projection directions (i.e. the surface normals) continuously vary over $\partial\mathcal{M}$ with respect to an infinitesimal change of $\mathbf{o}$. Therefore, we set $\mathbf{c}$ to $\mathbf{q}_i$ for every iteration.

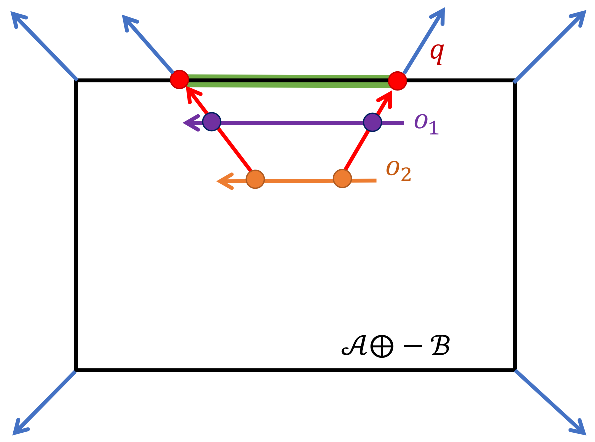


Fig. 10. The purple and orange lines represent two moving origins $\mathbf{o}_1$ and $\mathbf{o}_2$, respectively. Even though $\mathbf{o}_2$ moves only slightly compared to $\mathbf{o}_1$, their Phong-projected results on the surface of the Minkowski sum occupies the same region (the green line). Thus, it is necessary to have an expanded search region for a deeply-penetrating configuration. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Now we explain how to determine the radius r of S. We choose r to be proportional the magnitude of $\mathbf{q}_i$:

$$r = \alpha \|\mathbf{q}_i\|, \tag{9}$$

where α is a small constant in $\mathbb{R}^+$. The reasoning behind this choice is that deep penetration requires a wider search region on $\partial\mathcal{M}$, as illustrated in Fig. 10.

If a CPD solution is not found within S and $\mathbf{q}_i$ is not improving, we resize the radius of S by some constant factor of $\beta (> 1)$ and continue the search. The actual search process using Phong projection will be explained next.

6.2. *CPD computation using Phong projection*

Given an ϵ-ball S, we dynamically compute the surface of Minkowski sums $\partial\mathcal{M}(S)$ using the techniques explained in Secs. 4 and 5. Then, we test whether there is any triangle $T \in \partial\mathcal{M}(S)$ satisfying Eq. (4). If we find $\mathbf{q}$ satisfying Eq. (8), we stop the search and return $\mathbf{q}$ as CPD. Otherwise, we expand the search region S as explained in Section 6.1 and continue the search for a solution.

More specifically, if we cannot find a solution in $\partial\mathcal{M}(S_i)$, after we expand the search region from S_i to S_{i+1}, we find the nearest triangle in $\partial\mathcal{M}(S_i)$ from $\mathbf{q}_i$, extract the contact features (vertices, edges, facets) associated with $\mathbf{q}_i$, and reconstruct a Minkowski triangle. Then, we reconstruct a Minkowski surface connected by the triangle in breadth-first manner and perform Phong projection onto the surface.

Furthermore, when we initialize $\mathbf{q}_0$, we can either use the result of a conventional PD algorithm or simply reuse the result from the previous simulation frame. In the latter case, care should be taken as $\mathbf{q}_0$ from the previous simulation frame may not be collision-free any more at the current frame. In this case, the Minkowski triangle corresponding to $\mathbf{q}_0$ may not be on $\partial\mathcal{M}$, and thus we have no choice but to search all triangles in $\partial\mathcal{M}(S_0)$ to find a valid Phong projection.

Note that it is possible that no $\mathbf{q}$ may exist to satisfy Eq. (8) even if we construct the entire $\partial\mathcal{M}$, as a certain condition needs to be met for the existence of Phong projection; more details will be provided in the next section. In this case, we return $\mathbf{q}_i$ that best approximates Eq. (8) during the iterative search as CPD. The pseudocode of our CPD algorithm is summarized in Algorithm 2.

Algorithm 2 CPD Computation Algorithm

```
1: function CPD($\mathcal{A}$, $\mathcal{B}$)                    ▷ Two input models
2:   if motion coherence exists then
3:     $\mathbf{q}_0$ := CPD from the previous simulation frame
4:   else
5:     $\mathbf{q}_0$ := PolyDepth($\mathcal{A}$, $\mathcal{B}$)
6:   $r := \alpha\|\mathbf{q}_0\|$
7:   for $i := 0$ to max iterations do
8:     $\partial\mathcal{M}(S_i)$ := DynamicMinkowskiSum($S(\mathbf{q}_i, r)$)
9:     Compute $\Psi$ for $\partial\mathcal{M}(S_i)$
10:    $t$ := a triangle in $\partial\mathcal{M}(S_i)$ nearest from $\mathbf{q}_i$
11:    if $t \in$ contact surface then
12:      $\mathbf{q}$ := PhongPrjUsingBFS($\mathbf{o}$, $t$)
13:    else
14:      $\mathbf{q}$ := PhongPrj($\mathbf{o}$, $\partial\mathcal{M}(S)$)
15:    if $\mathbf{q}$ satisfies Def. 1 then return $\mathbf{q}$
16:    else
17:      $\mathbf{c}_{min} = \underset{\forall t \in \partial\mathcal{M}(S_i)}{\operatorname{argmin}} \{\Psi \cdot \text{(barycenter of } t)\}$
18:      if $\mathbf{q}_i = \mathbf{c}_{min}$ then
19:        $r := \beta r$
20:      else
21:        $\mathbf{q}_{i+1} := \mathbf{c}_{min}$
22:        $r := \alpha\|\mathbf{q}_{i+1}\|$
23:  return $\mathbf{q}_i$
```

Computational complexity. The worst-case computational complexity of our CPD algorithm is dominated by that of Minkowski-sum computation, multiplied by the number of iterations, as the Phong projection is a constant operation for each triangle.

6.3. *Continuity of Phong projection*

In general, if normals are defined continuously over a surface, Phong projections are continuous under the following assumption of dense tessellation.

Theorem 3. *Given a piecewise continuous surface $\mathcal{S}$, if for every triangle in $\mathcal{S}$ and the tangent planes $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$ of vertices on the triangle, $d(\mathbf{T}_i, \mathbf{T}_j) < \frac{1}{\sqrt{33}}$, where $d(\cdot)$ is a distance function using the Frobenius norm, then a Phong projection of a point close to $\mathcal{S}$ exists and is continuous* [32].

Our CPD solution is continuous when the underlying model $\mathcal{A}$ does not rotate, as its Minkowski sums are fixed and the normal vectors can be interpolated continuously over the surface of the Minkowski sum. When $\mathcal{A}$ rotates, the corresponding Minkowski sum also changes. The Minkowski sums under orientation change $\mathcal{M}_g$ (i.e. configuration space for $SE(3)$) can be considered as a stack of Minkowski sums with varying orientations of $\mathcal{M}$. Intuitively speaking, Phong projection should be continuous on $\mathcal{M}_g$ as the surface normals in $\mathcal{M}_g$ vary continuously. We prove this continuity property in the following theorem.

Theorem 4. *The Phong projection from the origin to $\partial\mathcal{M}$ is continuous when the orientation of $\mathcal{A}$ changes continuously.*

Proof. Let $\mathcal{M}(\varphi, \theta, \psi)$ be the Minkowski sums between $\mathcal{A}$ and $\mathcal{B}$ when $\mathcal{A}$ is rotated by $(\varphi, \theta, \psi) \in SO(3)$. Assume that the Phong projection of the origin $\mathbf{o} \in \mathbb{R}^3$ onto $\partial\mathcal{M}(\varphi, \theta, \psi)$ is a point $\mathbf{q} = (x, y, z) \in \mathbb{R}^3$ and its corresponding normal $\mathbf{n} = (n_1, n_2, n_3) \in \mathbb{R}^3$ such that:

$$(\mathbf{q} - \mathbf{o}) \times \mathbf{n} = \mathbf{0}, \tag{10}$$

and $\mathbf{q}$ is continuous as it is a Phong projection of $\mathbf{o}$ to $\partial\mathcal{M}(\varphi, \theta, \psi)$ in $\mathbf{R}^3$.

Now we pick a point $\mathbf{q}_g = (x, y, z, \varphi, \theta, \psi) \in \mathbb{R}^3 \times SO(3)$ from $\partial\mathcal{M}_g$, and set its corresponding normal to $\mathbf{n}_g = (n_1, n_2, n_3, 0, 0, 0)$ since $\mathbf{q}_g \cdot \mathbf{n}_g = 0$. Then, the Phong projection of the continuously moving origin $\mathbf{o}_g = (0, 0, 0, \varphi, \theta, \psi)$ in $\mathcal{M}_g$ is $\mathbf{q}_g = (x, y, z, \varphi, \theta, \psi)$ because

$$(\mathbf{q}_g - \mathbf{o}_g) \times \mathbf{n}_g = (\mathbf{q} - \mathbf{o}) \times \mathbf{n} = \mathbf{0}, \tag{11}$$

and $\mathbf{q}_g$ is continuous since $\mathbf{q}$ is continuous. □

7. Results and discussions

Now, we describe our experimental results of CPD computation and discuss them in comparison with other known algorithm.

7.1. Implementation and benchmarks

We have implemented our algorithm using C++ on a Windows 8, 64 bit operating system equipped with an Intel i7 3.60 GHz CPU and 16 GB memory. We use the PolyDepth library [15] to compute the conventional PD that was used to initialize our CPD algorithm, and the results of PolyDepth were also compared to our CPD results. We also use [32] as part of our Phong projection implementation. We set α and β in Algorithm 2 to 0.3 and 2, respectively, which gives the best results in our implementation.

We evaluated our CPD algorithm using the following benchmarks with different types of objects whose complexity ranges from 36 to 2K triangles, as shown in Fig. 11. In these benchmarks, we have a movable red object $\mathcal{A}$ and a fixed yellow object $\mathcal{B}$ in $\mathbb{R}^3$. As $\mathcal{A}$ moves along the blue arrow while colliding with $\mathcal{B}$, we compute their CPD and track the direction (x, y, z) as well as the magnitude of the CPD results. In particular, we display the CPD direction on a unit sphere (i.e. S^2).

We used the following benchmarking scenarios to analyze our CPD algorithm. We measure the computational performance of our CPD algorithm and show it in Table 1 by breaking down the timing into Minkowski sum computation and Phong projection.

- *Cone and axes*: The combinatorial complexities of the cone and axes model are 1K and 36, respectively. The cone translates to the left while rotating anti-clockwise. Our CPD results are clearly more continuous than PolyDepth, in particularly along the x direction. This corresponds to a large jump both in the direction and magnitude of PD results by PolyDepth.
- *Spoon and cup*: The spoon and cup models consist of 1.3 K and 1 K triangles, respectively. The spoon model rotates anti-clockwise around its local axis. Our CPD results are continuous while the PolyDepth results suddenly jump from inside to outside the cup or vice versa.
- *Fish and torus*: The fish and torus models consist of 950 and 1.6K triangles, respectively. The fish model rotates around its local axis. As the fish model rotates, the CPD results also rotate as expected while the PolyDepth shows a rather random behavior.
- *Torus and torus*: Each torus model has 2K triangles with a hole. One torus rotates around the other torus while being interlocked with each other. In this case, both CPD and PolyDepth show similar continuous results, but PolyDepth is still discontinuous near the beginning and end of simulation.

In all of these benchmarks, it is clear that our CPD algorithm generates continuous penetration depths while the conventional PD algorithm such as PolyDepth often shows discontinuous behaviors.

Dynamic Minkowski sum. We further compare the proposed dynamic Minkowski sum methods using bounding sphere hierarchies and gap filling to the method that computes the full Minkowski sum and a brute force method that computes all the convolution facets without using the convex map and uses only the ϵ-ball to filter the facets outside the ball. Results are reported in Table 2. The radius of the local ϵ-ball used in these experiment is 50. The smallest bounding spheres of the full Minkowski sums of all examples have radii either equal to or less than 250 in all examples. Example outputs of the full and local dynamic Minkowski sums are shown in Fig. 12.

Table 1
Average CPD timing performance in *ms*. The first column represents the benchmarking type in Fig. 11. The second to fourth columns represent the average timing for Minkowski sum computation, Phong projection and the total timing, respectively.

Type	Mink. Sum	Phong Prj.	Total
Cone/axes	0.35	0	1.39
Spoon/cup	1.35	0.03	4.23
Fish/torus	1.16	0	3.33
Torus/torus	3.18	0	4.42

Table 2
Average Minkowski computation time in *ms*. The second to fourth columns represent the average running time of computing the full Minkowski sum, local Minkowski sums using brute force, bounding sphere hierarchy and the gap filling approaches. The radius of the ϵ-ball is 50. See Fig. 12.

Type	Full	Brute F.	BSH	Gap Fill.
Cone/axes	58.91	16.10	24.23	10.33
Spoon/cup	1393.61	453.29	69.38	13.91
Fish/torus	1719.12	584.39	185.82	28.80
Torus/torus	2842.93	1158.74	390.65	36.06

Table 2 clearly shows that the computation times all reduce when the computation is confined to the ϵ-ball even when a brute-force approach is used. However, the difference is smaller when the inputs are more complex, e.g. torus/torus. Bounding sphere hierarchy and gap filling approaches provide further performance improvements. The improvement is particularly noticeable for larger models. In spoon/cup, fish/torus, and torus/torus cases, the gap filling approach is more than an order of magnitude faster than the brute-force methods and is always faster than the method using solely the bounding sphere hierarchy.

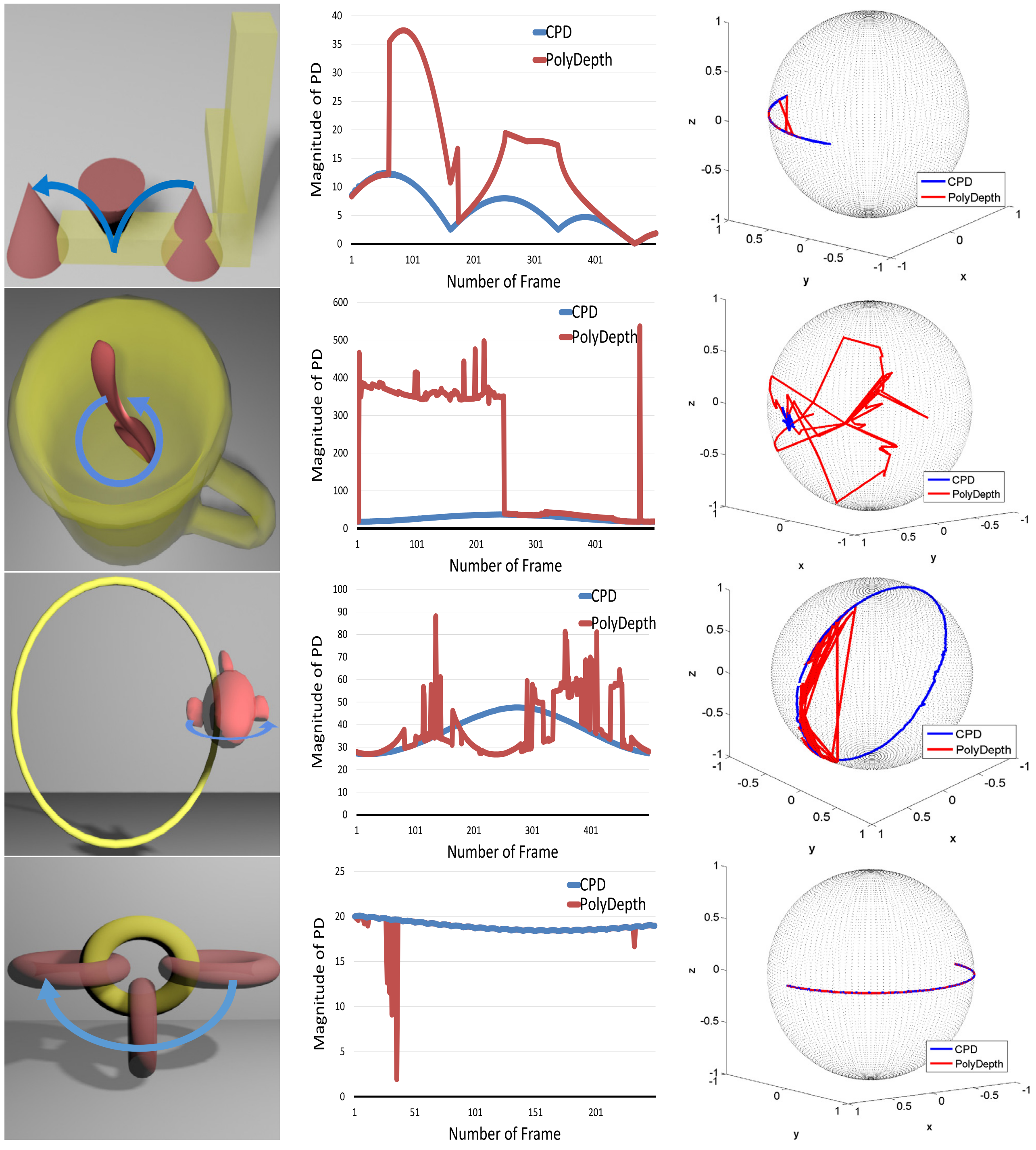
7.2. Discussions

Overall, our CPD algorithm not only generates a continuous PD, but also a smoother PD than a known PD method based on Euclidean projection. It also works for objects with a hole, as demonstrated in the fish/torus and torus/torus benchmarks. More interestingly, the magnitude of our CPD is often more optimal than that of PolyDepth in the cone/axes and spoon/cup benchmarks, as PolyDepth relies on a local projection technique without actually computing the surface of the Minkowski sum, whereas ours reconstructs the surface of the Minkowski sums. In other words, the projection technique used by PolyDepth may miss local details on the Minkowski sums surface.

In our experiments, the PD result computed by Euclidean projection (i.e. PolyDepth) is similar to that based on Phong projection. Thus, the Euclidean projection provides a good initializer for CPD computation. However, in theory, the PD result based on Euclidean projection can be different from that of CPD, when the surface of configuration space exhibits higher curvature with wild normal variation.

We would also like to emphasize that the proposed method works with other definitions of convolution, such as the idea of tracking [35,38], the only change that we need to make will be some details of convex map (i.e., phantom arcs) but the concept of convex map and the rest of the algorithms remain the same.

Limitations. Our CPD algorithm has a few limitations. Theoretically, it can guarantee gradient-continuity only if Theorem 3 is satisfied.



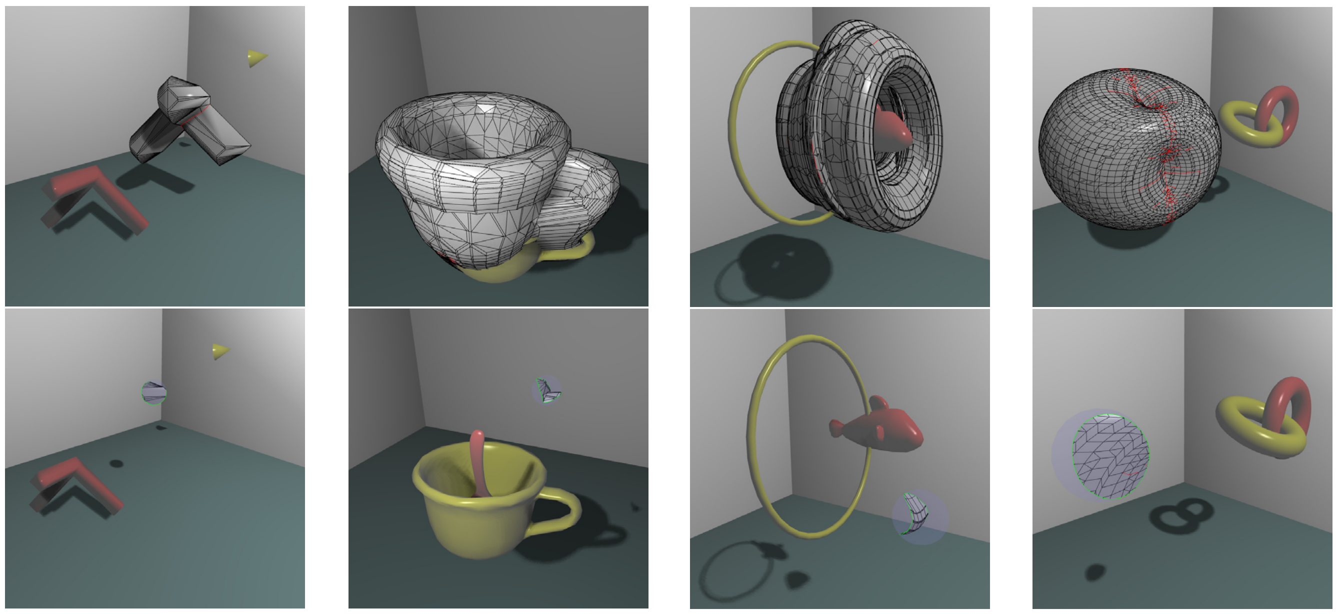
(a) Benchmarking Models. (b) PD magnitude. (c) PD direction.

Fig. 11. Various CPD benchmarks. (a) From top to bottom: cone/axes, spoon/cup, fish/torus, torus/torus. In each figure, the red model moves along the blue arrow while penetrating into the yellow model (b) PD magnitudes of our CPD and PD by PolyDepth. (c) PD directions of our CPD and PD by PolyDepth.

However, this condition is overly conservative in practice and, in our experiments, the results tend to be continuous. For a deep penetration case, the CPD solution may not be continuous either, as the solution space becomes narrow, bounded by normal variations near the origin. Finally, our algorithm may take more computational time when motion coherence is unavailable to exploit.

8. Conclusion

In this paper, we have presented a new algorithm to compute continuous PD at interactive rates. We estimate a solution subspace for CPD, and rapidly build local Minkowski sums in the subspace. Then, we interpolate tangent planes continuously over the boundary on the Minkowski sums and perform Phong projection onto it. Our algorithm is applicable to an arbitrary polygonal model with holes. For future work, we would like to further explore a CPD method that can handle both deep and shallow penetrations. We are also interested in finding a less conservative continuity condition than Theorem 3. Extending our algorithm to generalized PD is an interesting research direction and quite possible. Finally, we would like to apply our algorithm



(a) Cone/axes. (b) Spoon/cup. (c) Fish/torus. (d) Torus/torus.

Fig. 12. Top: full Minkowski sum. Bottom: local Minkowski sum. The radius of the local ϵ-ball is 50. The smallest bounding spheres of the full Minkowski sums have radii equal to or smaller than 250 in all four examples.

to applications such as penalty-based contact dynamics or haptic rendering to stabilize the simulation results.

Acknowledgments

This work was supported in part by NRF in Korea (2014K1A3A1A17073365, 2015R1A2A1A15055470). E. Behar and J.-M. Lien were supported by the Korean Federation of Science and Technology Societies (KOFST) grant funded by the Korean government, and by US NSF IIS-096053, CNS-1205260, EFRI-1240459, AFOSR FA9550-12-1-0238.

Appendix A

Details in dynamic convolution

Now that we know that we can construct, efficiently, the convex map of a non-convex polyhedron P, we can use it to dynamically construct the convolution $P \otimes Q$ of two arbitrary polyhedra. First we observe that each face $\overline{f} \in \mathcal{C}(P)$ represents a (possibly empty) set S which is a subset of the faces in P which have the same outward normal direction as $\overline{f}$, we call this set $\overline{F}$ the *correspondence set* of $\overline{f}$. Similarly, each edge $\overline{e}$ and vertex $\overline{v}$ represents a possibly empty set which is a subset of the edges or vertices, respectively, of P. We denote these correspondence sets $\overline{E}$ and $\overline{V}$ respectively. By the duality of the $\mathcal{C}(P)$ and $G(P)$, the following things are true:

1. Given any vertex $\overline{v} \in \mathcal{C}(P)$, for every $v \in \overline{V}$, $G(\overline{v})$ is wholly contained within $G(v)$, that is, $\forall \overline{v} \in \mathcal{C}(P), \forall v \in \overline{V} : G(\overline{v}) \subseteq G(v)$.
2. Given any edge $\overline{e} \in \mathcal{C}(P)$, for every $e \in \overline{E}$, $G(\overline{e})$ is wholly contained within $G(e)$ ($G(\overline{e}) \subseteq G(e)$).
3. Given any face $\overline{f} \in \mathcal{C}(P)$, for every $f \in \overline{F}$, $G(\overline{f}) = G(f)$.

As a consequence, given $\mathcal{C}(P)$ and $\mathcal{C}(Q)$, we have that if $\overline{f} \in \mathcal{C}(P)$ and $\overline{v} \in \mathcal{C}(Q)$ are compatible, then $\forall f \in \overline{F}, v \in \overline{V}, f$ is compatible with v. Similarly, for $\overline{e}_P \in \mathcal{C}(P), \overline{e}_Q \in \mathcal{C}(Q), \forall e_P \in \overline{E}_P, e_Q \in \overline{E}_Q$, we have that e_P is compatible with e_Q.

This correspondence between the labels of $\mathcal{C}(P) \otimes \mathcal{C}(Q)$ and $P \otimes Q$ enables us to use the techniques from [29] to update the convolution dynamically as Q rotates. First, we note that as Q rotates, so does $\mathcal{C}(Q)$–this is a trivial observation since the outward normals of $f \in F_Q$ rotate the same as the outward normals of $f \in \overline{F}_Q$. Let P after some arbitrary rotation be denoted $R(P)$.

A.1. Correcting vf-errors

We recall from [29] that a vf-error is an error arising in a face of the convolution which is contributed by a vertex from P and a face from Q, or vice-versa.

We can compute the vf-errors in compatibility between $R(\mathcal{C}(P))$ and $\mathcal{C}(Q)$ using the algorithm described in [29] with minor modifications. First, we compute $\mathcal{C}(P) \otimes \mathcal{C}(Q)$. Then, whenever we rotate P to some $R(P)$, we correct $\mathcal{C}(P) \otimes \mathcal{C}(Q)$. To correct $P \otimes Q$, assume that $\overline{f}_P$ and $\overline{v}_{Q1}$ are compatible in $\mathcal{C}(P) \otimes \mathcal{C}(Q)$. When $\overline{f}_P$ becomes incompatible with $\overline{v}_{Q1}$, it must then become compatible instead with some $\overline{v}_{Q2}$,[1] whose correspondence sets are $\overline{V}_{Q1}$ and $\overline{V}_{Q2}$, respectively.

After this happens, the faces in $\overline{F}_P$ are no longer compatible with the vertices in $\overline{V}_{Q1} - \overline{V}_{Q2}$, and are newly compatible with the faces in $\overline{V}_{Q2} - \overline{V}_{Q1}$. If $\overline{V}_{Q1} = \overline{V}_{Q2}$ then there is no error. This holds even when one or more of the correspondence sets are empty. Thus we can correct the convolution by removing the compatibilities in $\overline{V}_{Q1} - \overline{V}_{Q2}$, convolving every $f \in \overline{F}_P$ with every $v \in (\overline{V}_{Q2} - \overline{V}_{Q1})$, and updating the remaining faces $\overline{V}_{Q1} \cap \overline{V}_{Q2}$ for every $\overline{f} \in \mathcal{C}(P)$. The same argument applies in to faces from $\mathcal{C}(Q)$ and vertices from $\mathcal{C}(P)$–we may correct all vf-errors in $P \otimes Q$ using this method.

A.2. Correct ee-errors

We again recall from [29] that *ee*-errors arise in faces contributed to the convolution from compatibility between and edge of P and an edge of Q. To correct *ee*-errors, we discover *ee*-errors in $\mathcal{C}(P) \otimes \mathcal{C}(Q)$ using DYMSUM, and then correct them using

[1] As a reminder, convex polyhedra have the property that every region in the Gauss map arrangement is associated with precisely one vertex.

the label information similar to the vf-error case in Appendix A.1. Let $\overline{e}_P$ be compatible with a set of edges $\mathcal{E} = \overline{e}_{Q1}, \overline{e}_{Q2}, \ldots$.

After rotation, it instead is compatible with $\mathcal{E}' = \overline{e}'_{Q1}, \overline{e}'_{Q2}, \ldots$. Let $E_{old} = \cup_{\overline{e} \in \mathcal{E}} \overline{E}$ and $E_{new} = \cup_{\overline{e}' \in \mathcal{E}'} \overline{E}'$. Then for every $e_P \in \overline{E}_P$, it is no longer compatible with any $e_Q \in (E_{old} - E_{new})$, and it is newly compatible with every $e_Q \in (E_{new} - E_{old})$.

Once this is done, we have corrected the convolution.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cad.2016.05.012.

References

[1] Cameron S, Culley R. Determining the minimum translational distance between two convex polyhedra. In: Proceedings of international conference on robotics and automation, vol. 3, 1986, p. 591–6.
[2] Dobkin D, Hershberger J, Kirkpatrick D, Suri S. Computing the intersection-depth of polyhedra. Algorithmica 1993;9:518–33.
[3] Benson RV. Euclidean geometry and convexity. New York, NY: McGraw-Hill; 1966.
[4] Cameron S. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. In: Proceedings of international conference on robotics and automation, 1997, p. 3112–7.
[5] Attali D, Boissonnat J-D, Edelsbrunner H. Stability and computation of medial axes — a state-of-the-art report. In: Mathematical foundations of scientific visualization, computer graphics, and massive data exploration. Springer; 2009. p. 109–25.
[6] Tang M, Manocha D, Otaduy MA, Tong R. Continuous penalty forces. ACM Trans Graph 2012;31:107:1–107:9.
[7] Zhang X, Kim YJ, Manocha D. Continuous penetration depth. Comput Aided Des 2014;46:3–13.
[8] Lee Y, Kim Y. Phongpd: Gradient-continuous penetration metric for polygonal models using phong projection. In: Proceedings of international conference on robotics and automation, 2015, p. 57–62.
[9] Agarwal PK, Guibas LJ, Har-peled S, Rabinovitch A, Sharir M. Penetration depth of two convex polytopes in 3d. Nordic J Comput 2000;7:227–40.
[10] Van Den Bergen G. Proximity queries and penetration depth computation on 3d game objects. In: Game developers conference, vol. 170, 2001.
[11] Kim YJ, Lin MC, Manocha D. Incremental penetration depth estimation between convex polytopes using dual-space expansion. IEEE Trans Vis Comput Graphics 2004;10(2):152–63.
[12] Hachenberger P. Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. Algorithmica 2009;55(2):329–45.
[13] Kim YJ, Otaduy MA, Lin MC, Manocha D. Fast penetration depth computation for physically-based animation. In: ACM SIGGRAPH/eurographics symposium on computer animation, 2002, p. 23–31.
[14] Lien J-M. A simple method for computing minkowski sum boundary in 3d using collision detection. In: Algorithmic foundation of robotics VIII. Springer tracts in advanced robotics, vol. 57. 2009. p. 401–15.
[15] Je C, Tang M, Lee Y, Lee M, Kim YJ. Polydepth: Real-time penetration depth computation using iterative contact-space projection. ACM Trans Graph 2012;31(1):5:1–5:14.
[16] Zhang L, Kim YJ, Manocha D. A fast and practical algorithm for generalized penetration depth computation. In: Proceedings of robotics: science and systems conference, 2007.
[17] Nawratil G, Pottmann H, Ravani B. Generalized penetration depth computation based on kinematical geometry. Comput Aided Geom Design 2009;26(4):425–43.
[18] Tang M, Kim YJ. Interactive generalized penetration depth computation for rigid and articulated models using object norm. ACM Trans Graph 2014;33(1):1:1–1:15.
[19] Pan J, Zhang X, Manocha D. Efficient penetration depth approximation using active learning. ACM Trans Graph 2013;32(6):191:1–191:12.
[20] Keerthi S, Sridharan K. Measures of intensity of collision between convex objects and their efficient computation, The International Society for Optical Engineering, 1991, p. 266-75.
[21] Weller R, Zachmann G. Inner sphere trees for proximity and penetration queries. In: Robotics: science and systems, vol. 2. 2009.
[22] Allard J, Faure F, Courtecuisse H, Falipou F, Duriez C, Kry PG. Volume contact constraints at arbitrary resolution. In: Proceedings of SIGGRAPH 2010. ACM Trans Graph 2010;29(4):82:1–82:10.
[23] Tang M, Lee M, Kim YJ. Interactive hausdorff distance computation for general polygonal models. In: Proceedings of SIGGRAPH 2009. ACM Trans Graph 2009;28(3):74:1–74:9.
[24] Avnaim F, Boissonnat J. Polygon placement under translation and rotation. In: STACS, vol. 88. 1988. p. 322–33.
[25] Brost R. Computing metric and topological properties of configuration-space obstacles. In: 1989 IEEE international conference on robotics and automation, 1989. Proceedings., 1989, p. 170–6.
[26] Behar E, Lien J-M. Mapping the configuration space of polygons using reduced convolution. In: IROS. IEEE; 2013. p. 1242–8.
[27] Donald BR. A search algorithm for motion planning with six degrees of freedom. Artif. Intell. 1987;31(3):295–353.
[28] Mayer N, Fogel E, Halperin D. Fast and robust retrieval of minkowski sums of rotating polytopes in 3-space. Comput. Aided Des. 2011;43(10):1258–69.
[29] Behar E, Lien J-M. Dynamic Minkowski sum of convex shapes. In: Proc. of IEEE int. conf. on robotics and auteomation, 2011.
[30] Behar E, Lien J-M. Dynamic minkowski sums under scaling. Comput Aided Des. 2013;45(2):331–41.
[31] Phong BT. Illumination for computer generated pictures. Commun ACM 1975;18(6):311–7.
[32] Panozzo D, Baran I, Diamanti O, Sorkine-Hornung O. Weighted averages on surfaces. In: Proceedings of ACM SIGGRAPH. ACM Trans Graph 2013;32(4):60:1–60:12.
[33] Kobbelt L, Vorsatz J, Seidel H-P. Multiresolution hierarchies on unstructured triangle meshes. Computational Geometry: Theory and Applications 1999;14(1–3):5–24. Special issue on multi-resolution modelling and 3D geometry compression.
[34] Ghosh PK. A unified computational framework for Minkowski operations. Comput Graph 1993;17(4):357–78.
[35] Basch J, Guibas LJ, Ramkumar G, Ramshaw L. Polyhedral tracings and their convolution, 1996, p. 171–84.
[36] Wein R. Exact and efficient construction of planar Minkowski sums using the convolution method. In: Proc. 14th annual european symposium on algorithms, 2006, p. 829–40.
[37] Behar E, Lien J-M. Fast and robust 2d Minkowski sum using reduced convolution. In: Proc. IEEE int. conf. intel. rob. syst. (IROS), 2011.
[38] Ramkumar GD. Tracings and their convolution: theory and applications (Ph.D. thesis), stanford university; 1998.