C^{5}D: Sequential Continuous Convex Collision Detection Using Cone Casting

XIAODI YUAN, University of California San Diego, USA
FANBO XIANG, Hillbot Inc., USA
YIN YANG, University of Utah, USA
HAO SU, University of California San Diego, USA and Hillbot Inc., USA

Fig. 1. **Gachapon**: We simulate the interaction between nine free spheres and a rotating dispenser (silver) driven by a driving gear (orange) using ABD. Throughout the simulation, the non-intersection property is enforced by our proposed CCD algorithm for convex shapes undergoing affine motions. During time steps with peak complexity, our method handles 10 global CCD queries in 16.8 ms, achieving a 12× speed-up than the primitive-based ACCD baseline.

In physics-based simulation of rigid or nearly rigid objects, collisions often become the primary performance bottleneck, particularly when enforcing intersection-free constraints. Previous simulation frameworks rely on primitive-level CCD algorithms. Due to the large number of colliding surface primitives to process, those methods are computationally intensive and heavily dependent on advanced parallel computing resources such as GPUs, which are often inaccessible due to competing tasks or capped threading capacity in applications like policy training for robotics. To address these limitations, we propose a sequential CCD algorithm for convex shapes undergoing constant affine motion. This approach uses the conservative advancement method to iteratively refine a lower-bound estimate of the TOI, exploiting the linearity of affine motion and the efficiency of convex shape distance computation. Our CCD algorithm integrates seamlessly into the ABD framework, achieving a 10-fold speed-up over primitive-level CCD. Its high single-threaded efficiency further enables significant throughput improvements via scene-level parallelism, making it well-suited for resource-constrained environments.

Authors' addresses: Xiaodi Yuan, University of California San Diego, La Jolla, CA, USA, x9yuan@ucsd.edu; Fanbo Xiang, Hillbot Inc., La Jolla, CA, USA, fx@hillbot.ai; Yin Yang, University of Utah, Salt Lake City, UT, USA, yin.yang@utah.edu; Hao Su, University of California San Diego, 9500 Gilman Dr, La Jolla, CA, USA and Hillbot Inc., La Jolla, CA, USA, haosu@ucsd.edu.

ACM 0730-0301/2025/8-ART
https://doi.org/10.1145/3731151

CCS Concepts: • **Computing methodologies** → **Collision detection**; **Physical simulation**.

Additional Key Words and Phrases: Continuous collision detection, Rigid body dynamics

ACM Reference Format:
Xiaodi Yuan, Fanbo Xiang, Yin Yang, and Hao Su. 2025. C^{5}D: Sequential Continuous Convex Collision Detection Using Cone Casting. *ACM Trans. Graph.* 44, 4 (August 2025), 14 pages. https://doi.org/10.1145/3731151

1 INTRODUCTION

The simulation of rigid or nearly rigid objects is a ubiquitous task in computer graphics and robotics. As highly stiff materials resist body deformation strongly, a common practice is to use a set of reduced coordinates to simplify its dynamics i.e., the classic rigid body dynamics (RBD) [Baraff 1997], which results in order-of-magnitude speed-ups compared with FEM-based methods [Bathe 2006]. On the other hand, processing interactions induced by collisions among stiff bodies emerges as a critical challenge, particularly when intersection-free collision needs to be strictly enforced.

Incremental potential contact (IPC) [Li et al. 2020] offers a promising solution for collision and contact handling and has been proven effective for RBD [Ferguson et al. 2021]. Unfortunately, as classic RBD yields a curly trajectory between time steps, IPC for RBD is costly for calculating the time of impact (TOI). Lan et al. [2022] proposed the affine body dynamics (ABD) to obviate this limitation.

Unlike RBD, ABD relaxes rigidity to allow small deformation in an affine subspace, leading to piecewise linear motion so that the regular continuous collision detection (CCD) method remains valid.

Nevertheless, those previous methods, such as IPC and ABD, rely on *primitive-level* CCD algorithms that compute the TOI via solving cubic polynomials, either analytically [Li et al. 2020] or numerically [Lan et al. 2022; Li et al. 2021]. The performance of these algorithms is up to the number of primitives. Even after broad-phase culling with spatial data structures such as hash grids [Teschner et al. 2003] and bounding volume hierarchy (BVH) [Wang et al. 2018], complicated scenes often yield a large number of colliding pairs. Hardware-level parallelism becomes the only option for further improving the efficiency of collision processing.

Unfortunately, adequate hardware resources for massive parallelization are not always available. Many applications do not have access to high-end GPUs, and even when they do, these GPUs are also needed for other computing tasks such as rendering. This issue is even more prominent when training robotics policies in simulators, where thousands of robot instances are simulated in parallel. In this scenario, parallelism is trivially achieved across instances, and the overall simulation performance comes down to the single-threaded execution efficiency for each instance.

To address the challenge of CCD with limited computational resources, instead of handling numerous primitives, we decompose the collision shapes into a small number of *convex* proxies using approximate convex decomposition (ACD) [Wei et al. 2022] and perform shape-level CCD on these convex proxies—a pipeline widely adopted by real-time simulators such as Bullet [Coumans and Bai 2021] and MuJoCo [Todorov et al. 2012]. When applied to ABD, this convexity assumption introduces a challenge that, to the best of our knowledge, has not been previously addressed: CCD for convex shapes under affine motion. We propose a novel algorithm based on conservative advancement (CA) [Mirtich 1996], which iteratively refines the TOI estimate by advancing toward a lower-bound TOI derived from the current distance measurement. Leveraging the affine trajectories, our method achieves efficient convergence through tight TOI bounds and integrates seamlessly into the ABD framework.

Leveraging the small number of convex parts and the efficiency of CCD between them, our algorithm delivers high performance even in single-threaded implementations, achieving speeds approximately 10 times faster than the primitive-level ACCD [Li et al. 2021]. Its low computational resource requirements also enable efficient parallel simulation of multiple scenes, further increasing the overall throughput by 14 times on a multi-core CPU—a significant advantage for downstream applications such as robotics. Our implementation of the proposed CCD algorithms is available at https://github.com/Rabbit-Hu/c5d.

2 RELATED WORK

2.1 Stiff and Rigid Body Simulation

Rigid body simulation has been extensively studied in graphics and robotics [Baraff 1997; Bender et al. 2014; Mirtich and Canny 1995]. Since the configuration of a rigid object can be fully described with six variables, rigid body simulation puts more emphasis on how to handle collisions and contacts. Treating collisions as hard non-intersection constraints traditionally leads to algorithms based on linear complimentary programming (LCP) [Anitescu and Potra 1997; Baraff 1994, 1995; Erleben 2007; Kaufman et al. 2005; Stewart 2000; Trinkle et al. 2001], which is generally an NP-hard problem. In contrast, penalty-based algorithms [Drumwright 2007; Fisher and Lin 2001; Hasegawa et al. 2004; Macklin et al. 2016; Müller et al. 2007; Tang et al. 2014; Xu et al. 2014] often replace hard non-intersection constraints with soft, spring-like penalties. However, such penalties introduce the risk of unresolved intersections and inaccurate contact forces, restricting their use primarily to visually plausible simulations.

As a distinctive penalty-based approach, the recent incremental potential contact (IPC) algorithms [Li et al. 2020], such as Rigid-IPC [Ferguson et al. 2021] and ABD [Chen et al. 2022; Huang et al. 2024; Lan et al. 2022], employ barrier functions to enforce non-intersection. When combined with a CCD-filtered projected Newton solver, IPC guarantees robust and efficient intersection-free simulation. However, solving IPC-based systems is particularly CCD-intensive, as a global CCD must be performed in every Newton iteration. In contrast, classic rigid-body simulators such as Bullet [Coumans and Bai 2021] typically only use CCD to find potential contact pairs and avoid missing contacts between fast-moving objects.

From a CCD perspective, a particularly important feature of (nearly-)rigid body simulators is the nature of the piecewise *trajectories* produced by their iterative solvers. Classic methods such as Bullet directly formulate rigid body models with 6-DOF $SE(3)$ coordinates and iteratively update them by small time steps, during which the translational and linear velocities remain constant, hence generating a piecewise *screw* trajectory. In contrast, Rigid-IPC linearly interpolates between rotation vectors $\boldsymbol{\theta}$ before applying Rodrigues' formula, resulting in a piecewise curved, non-screw trajectory that requires a specialized curved CCD algorithm. Lastly, ABD relaxes the rigidity constraint by adopting 12-DOF affine coordinates, hence generating a piecewise linear trajectory where each segment is defined by constant affine velocities (see Section 3). This linear trajectory enables efficient linear CCD methods, which is a key factor behind ABD's significant speed-up over Rigid-IPC.

2.2 Continuous Collision Detection

Primitive-Level CCD. Primitive-level CCD methods compute the TOI between pairs of geometric primitives, typically vertex-triangle or edge-edge pairs. We refer the reader to [Wang et al. 2021] for a comprehensive survey on this class of methods. Using the closed-form expression of the primitive distance as a function of t, the TOI can be algebraically solved through polynomial equations applicable to both linear [Provot 1997] and screw motion [Canny 1986; Redon et al. 2000]. Although floating-point root-finding techniques are efficient, they are vulnerable to numerical errors, frequently resulting in false positives and negatives for closely positioned or degenerate primitive pairs. In contrast, conservative advancement (CA) methods iteratively refine a lower-bound estimate of the TOI using simple floating-point arithmetic, thereby enhancing robustness against numerical inaccuracies. Examples include ACCD [Li

et al. 2021] for linear motion, [Ferguson et al. 2021] for curved motion, and [Tang et al. 2009] for screw motion. When employing primitive-level methods to determine the TOI between triangular meshes, it is necessary to traverse all potential primitive pairs and select the minimum TOI, a process that can be computationally intensive for high-resolution meshes.

Shape-level CCD. Shape-level CCD methods process pairs of moving shapes holistically without decomposing them into numerous primitive pairs. They often require convexity of the collision shapes to facilitate efficient computation of distances and TOI bounds. van den Bergen and Gregorius [2010] transforms the CCD of convex shapes under affine motion to a CA-like ray-casting process onto a Minkowski sum. Bullet [Coumans and Bai 2021] extends this approach to screw motion by integrating angular velocities into the computation of lower-bound TOI estimates within CA iterations, building upon the framework of [Mirtich 1996]. In this paper, we propose to extend shape-level CA-based CCD to affine motions, leveraging the advantages of affine motions to obtain tighter TOI bounds and thereby accelerate convergence.

3 PROBLEM FORMULATION

In this paper, we focus on the problem of continuous collision detection between convex shapes that undergo piecewise affine motion. We start with the problem definition.

Continuous collision detection. Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be bounded and closed subsets of $\mathbb{R}^3$ representing the shapes of two 3D objects. Their specific configurations (e.g., positions and orientations) vary along the time t, i.e., $\mathcal{B}_1(t)$ and $\mathcal{B}_2(t)$. The continuous collision detection (CCD) seeks the time of impact (TOI), t^*, defined by:

$$t^* = t^*(\mathcal{B}_1, \mathcal{B}_2) = \min\{t \geq 0 \mid \mathcal{B}_1(t) \cap \mathcal{B}_2(t) \neq \emptyset\}. \tag{1}$$

To address the numerical instability of floating-point arithmetic, a common practice is to find a conservative estimation, $\hat{t}^* \in (0, t^*]$, which maintains a small gap between shapes:

$$s\bar{d} \leq d\big(\mathcal{B}_1(\hat{t}^*)), \mathcal{B}_2(\hat{t}^*)\big) < \alpha s\bar{d}, \tag{2}$$

where the distance function d is defined as:

$$d(\mathcal{B}_1, \mathcal{B}_2) = \min\{\|\boldsymbol{b}_1 - \boldsymbol{b}_2\| \mid \boldsymbol{b}_1 \in \mathcal{B}_1, \boldsymbol{b}_2 \in \mathcal{B}_2\}, \tag{3}$$

$\bar{d} = d(\mathcal{B}_1(0), \mathcal{B}_2(0))$, $s \in (0, 1)$ and $\alpha > 1$ are a small scaling factors, e.g., we can have $s = 0.01$ and $\alpha = 10$. $s\bar{d}$ serves as a minimal separation requirement that prevents potential numerical instability.

We start with the exact CCD formulation (Equation (1)) and show later (Section 4.3) that our proposed method can be easily modified to suit the conservative formulation of CCD (Equation (2)).

Constant affine motion. We focus on CCD between a pair of time-varying shapes $\mathcal{B}_1(t), \mathcal{B}_2(t)$, each undergoing constant affine motion:

$$\mathcal{B}(t) = (\boldsymbol{I} + t\boldsymbol{A})\bar{\mathcal{B}} \oplus \{t\boldsymbol{v}\} = \left\{\bar{\boldsymbol{b}} + t\boldsymbol{A}\bar{\boldsymbol{b}} + t\boldsymbol{v} \mid \bar{\boldsymbol{b}} \in \bar{\mathcal{B}}\right\}, \tag{4}$$

where $\bar{\mathcal{B}}$ represents the *rest shape*, $\boldsymbol{I}$ is the identity matrix, and the affine velocity consists of the linear velocity matrix $\boldsymbol{A}$ and the translational velocity vector $\boldsymbol{v}$. The notation $\oplus$ denotes the Minkowski sum [van den Bergen 2004] such that:

$$\mathcal{A} \oplus \mathcal{B} = \{\boldsymbol{a} + \boldsymbol{b} \mid \boldsymbol{a} \in \mathcal{A}, \boldsymbol{b} \in \mathcal{B}\}. \tag{5}$$

When $\mathcal{A}$ and $\mathcal{B}$ are both convex, $\mathcal{A} \oplus \mathcal{B}$ is also convex. Note that Equation (4) assumes trivial initial transformation, i.e., $\mathcal{B}(0) = \bar{\mathcal{B}}$. A more general affine motion with non-trivial initial transformation,

$$\mathcal{B}(t) = (\bar{\boldsymbol{A}} + t\boldsymbol{A})\bar{\mathcal{B}} \oplus \{\bar{\boldsymbol{v}} + t\boldsymbol{v}\}, \tag{6}$$

can be transformed to the form of Equation (4) as:

$$\mathcal{B}(t) = (\boldsymbol{I} + t\boldsymbol{A}\bar{\boldsymbol{A}}^{-1})\left(\bar{\boldsymbol{A}}\bar{\mathcal{B}} \oplus \{\bar{\boldsymbol{v}}\}\right) \oplus \left\{t(\boldsymbol{v} - \boldsymbol{A}\bar{\boldsymbol{A}}^{-1}\bar{\boldsymbol{v}})\right\}, \tag{7}$$

allowing general affine motions with arbitrary initial configurations to be analyzed under the simplified model in Equation (4). When $\bar{\mathcal{B}}_1, \bar{\mathcal{B}}_2$ are *convex*, the shape convexity is preserved under affine transformations, meaning $\mathcal{B}_1(t), \mathcal{B}_2(t)$ are also convex.

4 CONE-CASTING CCD (C^{5}D-LINEAR)

We now give the detail of a CA-like algorithm to solve the aforementioned CCD problem. At every iteration, our algorithm generates a safe estimate, $\hat{t}^* \in (0, t^*]$, of the actual TOI $t^* = t^*(\mathcal{B}_1, \mathcal{B}_2)$ and forwards the current time to $\hat{t}^*$. This procedure repeats until either t reaches $+\infty$ or a collision is detected, i.e., $d\big(\mathcal{B}_1(t), \mathcal{B}_2(t)\big) = 0$. In practice, around 60 iterations offer a sufficiently accurate estimate of the TOI. The computation of $\hat{t}^*$ is derived in Sections 4.1 and 4.2, while Section 4.3 discusses the transition between iterations.

4.1 Cone casting as a conservative estimation

We first focus on the computation of $\hat{t}^*$ in the first iteration. Consider two convex shapes under affine motions:

$$\mathcal{B}_1(t) = (\boldsymbol{I} + t\boldsymbol{A}_1)\bar{\mathcal{B}}_1 \oplus \{t\boldsymbol{v}_1\}, \;\; \mathcal{B}_2(t) = (\boldsymbol{I} + t\boldsymbol{A}_2)\bar{\mathcal{B}}_2 \oplus \{t\boldsymbol{v}_2\}. \tag{8}$$

Assuming the algorithm has not terminated, we have $d(\bar{\mathcal{B}}_1, \bar{\mathcal{B}}_2) > 0$. Similar to Equation (5), let $\ominus$ denote the Minkowski difference:

$$\mathcal{A} \ominus \mathcal{B} = \mathcal{A} \oplus (-\mathcal{B}) = \{\boldsymbol{a} - \boldsymbol{b} \mid \boldsymbol{a} \in \mathcal{A}, \boldsymbol{b} \in \mathcal{B}\}, \tag{9}$$

and $d(\mathcal{B}_1, \mathcal{B}_2)$ can be rewritten as $d(\mathcal{B}_1 \ominus \mathcal{B}_2, \{\boldsymbol{0}\})$. For simplicity, we use $d(\mathcal{B})$ as a shorthand for $d(\mathcal{B}, \{\boldsymbol{0}\})$, and define $\boldsymbol{c}(\mathcal{B}) = \operatorname{argmin}_{\boldsymbol{b} \in \mathcal{B}} \|\boldsymbol{b}\|$. It directly follows that $d(\mathcal{B}) = \|\boldsymbol{c}(\mathcal{B})\|$. Since $\mathcal{B}_1$ and $\mathcal{B}_2$ are both convex, $\boldsymbol{c}(\mathcal{B}_1 \ominus \mathcal{B}_2)$ and $d(\mathcal{B}_1 \ominus \mathcal{B}_2)$ can be efficiently computed using the GJK algorithm [Gilbert et al. 1988; Montanari et al. 2017]. These new notations allow three equivalent formulations of *"$\mathcal{B}_1$ collides with $\mathcal{B}_2$"*:

$$\mathcal{B}_1 \cap \mathcal{B}_2 \neq \emptyset \Leftrightarrow \boldsymbol{0} \in \mathcal{B}_1 \ominus \mathcal{B}_2 \Leftrightarrow d(\mathcal{B}_1 \ominus \mathcal{B}_2) = 0. \tag{10}$$

At time t, the Minkowski difference evolves into:

$$\begin{aligned} \mathcal{B}_1(t) \ominus \mathcal{B}_2(t) &= (\boldsymbol{I} + t\boldsymbol{A}_1)\bar{\mathcal{B}}_1 \ominus (\boldsymbol{I} + t\boldsymbol{A}_2)\bar{\mathcal{B}}_2 \ominus \{t\boldsymbol{v}_{12}\} \\ &\subseteq \bar{\mathcal{B}}_1 \oplus t\boldsymbol{A}_1\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2 \ominus t\boldsymbol{A}_2\bar{\mathcal{B}}_2 \ominus \{t\boldsymbol{v}_{12}\} \\ &= (\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t). \end{aligned} \tag{11}$$

Here, we have $\boldsymbol{v}_{12} = \boldsymbol{v}_2 - \boldsymbol{v}_1$, and $C(t) = t\bar{C}$ for $\bar{C} = \boldsymbol{A}_2\bar{\mathcal{B}}_2 \ominus \boldsymbol{A}_1\bar{\mathcal{B}}_1 \oplus \{\boldsymbol{v}_{12}\}$. This leads to $d\big(\mathcal{B}_1(t) \ominus \mathcal{B}_2(t)\big) \geq d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t)\big)$, and therefore:

$$\begin{aligned} t^*(\mathcal{B}_1, \mathcal{B}_2) &= \min\left\{t \geq 0 \mid d\big(\mathcal{B}_1(t) \ominus \mathcal{B}_2(t)\big) = 0\right\} && (12) \\ &\geq \min\left\{t \geq 0 \mid d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t)\big) = 0\right\} && (13) \\ &= t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big). && (14) \end{aligned}$$

Notice that $d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(0)\big) = d(\bar{\mathcal{B}}_1, \bar{\mathcal{B}}_2) > 0$, and thus $t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big) > 0$. We say $\hat{t}^* = t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big)$ is a *conservative estimate* of $t^*(\mathcal{B}_1, \mathcal{B}_2)$, which can be computed through the algorithm presented in Section 4.2.

Discussion. The TOI estimate $\hat{t}^* = t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big)$ can be interpreted as the earliest time t when the static shape $\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2$ is hit by a truncated cone $\{s\boldsymbol{x} \mid 0 \le s \le t, \boldsymbol{x} \in \bar{C}\}$ (see Figure 2(b)). We refer to this formulation as *cone casting*, a generalization to the widely used *ray casting* technique in rendering and game development. When both $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ become vanished, the cone casting reduces to ray casting, where the ray starts at $\boldsymbol{0}$ and travels along $\boldsymbol{v}_{12}$, and the inequality Equation (13) reaches its equality. In this case, the TOI can be obtained using the ray-casting CCD algorithm [van den Bergen and Gregorius 2010].

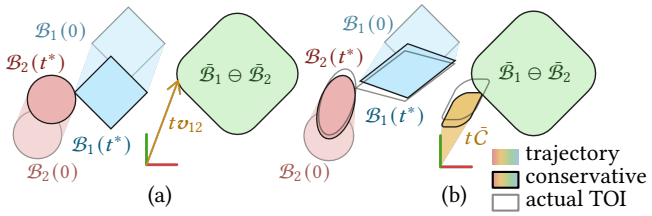

Fig. 2. **Ray casting and cone casting**: The ray and the truncated cone are shown in yellow. **(a)**: Constant translational motion and ray casting. Upon convergence, a single execution of the ray-casting algorithm computes $\hat{t}^* = t^*(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, \{t\boldsymbol{v}_{12}\}) = t^*(\mathcal{B}_1, \mathcal{B}_2)$. **(b)**: Constant affine motion and cone casting. Upon convergence, a single execution of the cone-casting algorithm computes $\hat{t}^* = t^*(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, t\bar{C}) \le t^*(\mathcal{B}_1, \mathcal{B}_2)$, where the LHS serves as a conservative estimate for the RHS.

4.2 Cone casting by conservative advancement

We show that the CA-based ray casting algorithm [van den Bergen and Gregorius 2010] can be conveniently generalized for cone casting problems, i.e., for computing $t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big)$.

Let $\boldsymbol{s}_{\mathcal{B}}(\boldsymbol{v})$ be the *support mapping* that maps a given $\boldsymbol{v}$ to a point in $\mathcal{B}$ such that $\boldsymbol{v} \cdot \boldsymbol{s}_{\mathcal{B}}(\boldsymbol{v}) = \max_{\boldsymbol{b} \in \mathcal{B}} \boldsymbol{v} \cdot \boldsymbol{b}$, or equivalently:

$$\boldsymbol{s}_{\mathcal{B}}(\boldsymbol{v}) \in \operatorname*{argmax}_{\boldsymbol{b} \in \mathcal{B}} \boldsymbol{b} \cdot \boldsymbol{v}. \tag{15}$$

This mapping holds the following properties [van den Bergen 2004]:

$$\boldsymbol{s}_{\mathcal{A} \oplus \mathcal{B}}(\boldsymbol{v}) = \boldsymbol{s}_{\mathcal{A}}(\boldsymbol{v}) + \boldsymbol{s}_{\mathcal{B}}(\boldsymbol{v}), \tag{16}$$

$$\boldsymbol{s}_{k\mathcal{B}}(\boldsymbol{v}) = k\boldsymbol{s}_{\mathcal{B}}(\boldsymbol{v}), \quad \forall k \ge 0, \tag{17}$$

$$\boldsymbol{s}_{-\mathcal{B}}(\boldsymbol{v}) = -\boldsymbol{s}_{\mathcal{B}}(-\boldsymbol{v}), \tag{18}$$

$$\boldsymbol{s}_{\boldsymbol{A}\mathcal{B}}(\boldsymbol{v}) = \boldsymbol{A}\boldsymbol{s}_{\mathcal{B}}(\boldsymbol{A}^\top \boldsymbol{v}). \tag{19}$$

For a closed convex set $\mathcal{B}$, the support vector of every unit vector $\hat{\boldsymbol{n}}$ gives a lower bound on $d(\mathcal{B})$:

$$d(\mathcal{B}) = \min_{\boldsymbol{b} \in \mathcal{B}} \|\boldsymbol{b}\| \ge \min_{\boldsymbol{b} \in \mathcal{B}} \boldsymbol{b} \cdot \hat{\boldsymbol{n}} = \boldsymbol{s}_{\mathcal{B}}(-\hat{\boldsymbol{n}}) \cdot \hat{\boldsymbol{n}}, \tag{20}$$

where the equality holds when $\hat{\boldsymbol{n}} = \boldsymbol{c}(\mathcal{B})/d(\mathcal{B})$ (see Theorem 1 in the appendix for details).

Substituting $\mathcal{B}$ with $(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t)$, Equation (20) can be further simplified to

$$\begin{aligned} &d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t)\big) = d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus t\bar{C}\big) \\ &\ge \boldsymbol{s}_{(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus t\bar{C}}(-\hat{\boldsymbol{n}}) \cdot \hat{\boldsymbol{n}} = \boldsymbol{s}_{\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2}(-\hat{\boldsymbol{n}}) \cdot \hat{\boldsymbol{n}} - t\boldsymbol{s}_{\bar{C}}(\hat{\boldsymbol{n}}) \cdot \hat{\boldsymbol{n}}. \end{aligned}$$

By setting $\hat{\boldsymbol{n}} = \hat{\boldsymbol{n}}^* = \boldsymbol{c}(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2)/d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2)$, we have:

$$d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t)\big) \ge d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) - tV. \tag{21}$$

It is noted that $V = \boldsymbol{s}_{\bar{C}}(\hat{\boldsymbol{n}}^*) \cdot \hat{\boldsymbol{n}}^*$, and it can be efficiently computed using Equation (19):

$$V = \Big(\boldsymbol{A}_2 \boldsymbol{s}_{\bar{\mathcal{B}}_2}(\boldsymbol{A}_2^\top \hat{\boldsymbol{n}}^*) - \boldsymbol{A}_1 \boldsymbol{s}_{\bar{\mathcal{B}}_1}(-\boldsymbol{A}_1^\top \hat{\boldsymbol{n}}^*) + \boldsymbol{v}_2 - \boldsymbol{v}_1\Big) \cdot \hat{\boldsymbol{n}}^*. \tag{22}$$

If $V \le 0$, we have that for any $t \ge 0$,

$$d\big(\mathcal{B}_1(t) \ominus \mathcal{B}_2(t)\big) \ge d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus t\bar{C}\big) \ge d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) > 0,$$

meaning shapes do not collide, or $t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big) = +\infty$. On the other hand, when $V > 0$, $t \le \hat{t}^*$ for $\hat{t}^* = d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2)/V > 0$ becomes a sufficient condition of $d\big((\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) \ominus C(t)\big) \ge 0$. We therefore argue that $\hat{t}^*$ is a conservative estimate of $t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big)$.

Note that $\hat{t}^*$ is also a conservative estimate of $t^*(\mathcal{B}_1, \mathcal{B}_2)$, the ultimate goal of the CCD algorithm. This implies that it is unnecessary (though possible) to run multiple *cone-casting CA iterations* to refine the estimate of $t^*\big(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2, C\big)$. A single cone-casting iteration is sufficient to find a $\hat{t}^* \le t^*(\mathcal{B}_1, \mathcal{B}_2)$, enabling the algorithm to advance the current time by $\hat{t}^*$ and proceed to the next *CCD CA iteration* for computing $t^*(\mathcal{B}_1, \mathcal{B}_2)$. The advancement alters the shape of $\bar{C}$, resulting in a new cone-casting problem in the subsequent CCD CA iteration, as discussed in detail in Section 4.3.

4.3 Advancement

As the last step in one CA iteration, the current time T is advanced to $T + \hat{t}^*$ computed in Section 4.2. For $i = 1, 2$ respectively, the motion *starting* from any time point T, $\mathcal{B}_i(T + t)$, can be transformed into the form defined in Equation (4):

$$\mathcal{B}_i(T + t) = \mathcal{B}_i'(t) = (\boldsymbol{I} + t\boldsymbol{A}_i')\bar{\mathcal{B}}_i' \oplus \{t\boldsymbol{v}_i'\}, \quad i = 1, 2, \tag{23}$$

where $\mathcal{B}_i', \boldsymbol{A}_i', \boldsymbol{v}_i'$ are computed by substituting $\bar{\boldsymbol{A}} = (\boldsymbol{I} + T\boldsymbol{A}_i), \bar{\boldsymbol{v}} = T\boldsymbol{v}_i$ into Equation (7) respectively for $i = 1, 2$. This transformation allows us to perform the same procedure as described in Sections 4.1 and 4.2 to obtain a new conservative estimate of $t^*(\mathcal{B}_1', \mathcal{B}_2')$. In every iteration, a positive $\hat{t}^*$ is obtained and leads us closer to the TOI, until convergence is achieved when $d(\mathcal{B}_1'(0), \mathcal{B}_1'(0)) = 0$. Convergence is guaranteed by the continuity and boundedness of the mapping $\hat{t}^* = t^*(T)$, for which a detailed proof is provided in the appendix (Section E).

When solving the conservative version of the CCD problem (Equation (2)), we advance the current time T by a smaller step $t_\alpha := \beta\hat{t}^*$ instead of $\hat{t}^*$ and continue the CA iterations until the new distance $d(\mathcal{B}_1(t_\alpha) \ominus \mathcal{B}_2(t_\alpha))$ becomes smaller than $s\bar{d}$. The step size β is set to $1 - s$ for the first iteration and $1 - \frac{1}{\alpha}$ for later iterations. In the appendix (Section D), we prove that when $d(\mathcal{B}_1(T + t_\alpha) \ominus \mathcal{B}_2(T + t_\alpha)) < s\bar{d}$ for the first time, the requirements of the conservative CCD problem are satisfied, i.e., $s\bar{d} \le d(\mathcal{B}_1(0), \mathcal{B}_2(0)) \le \alpha s\bar{d}$.

Algorithm 1 summarizes the first CCD algorithm presented in this paper, referred to as "C^{5}D-Linear", whose variants will be presented in Section 5.

Algorithm 1: C^{5}D-Linear

Procefure `Support`$(\mathcal{B}, \boldsymbol{v})$:
  Input: $\mathcal{B}$: a closed shape
    $\boldsymbol{v}$: vector
  Output: $\boldsymbol{s}$: The support vector $\boldsymbol{s}_{\mathcal{B}}(\boldsymbol{v})$
  ...
Procefure `GJK`$(\mathcal{B}_1, \mathcal{B}_2)$:
  Input: $\mathcal{B}_1, \mathcal{B}_2$: two convex shapes
  Output: d: the distance $d(\mathcal{B}_1 \ominus \mathcal{B}_2)$
    $\boldsymbol{c}$: the distance vector $\boldsymbol{c}(\mathcal{B}_1 \ominus \mathcal{B}_2)$
  ...
Procedure `C5D-Linear`$(\bar{\mathcal{B}}_1, \bar{\mathcal{B}}_2, \bar{A}_1, \bar{A}_2, \bar{\boldsymbol{v}}_1, \bar{\boldsymbol{v}}_2, \boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{v}_1, \boldsymbol{v}_2, s, \alpha)$:
  Input: $\bar{\mathcal{B}}_i, \bar{A}_i, \bar{\boldsymbol{v}}_i, \boldsymbol{A}_i, \boldsymbol{v}_i$: motion $\mathcal{B}_i(t) = (\bar{A}_i + t\boldsymbol{A}_i)\bar{\mathcal{B}}_i \oplus \{\bar{\boldsymbol{v}}_i + t\boldsymbol{v}_i\}$
    s, α: scaling factors required by conservative CCD
  Output: $\hat{T}^*$: The conservative TOI
  $d, \boldsymbol{c} \leftarrow$ `GJK` $\left(\mathcal{B}_1(0), \mathcal{B}_2(0)\right)$
  $\bar{d} \leftarrow d$
  $T \leftarrow 0$
  $\beta \leftarrow 1 - s$
  while *true* **do**
    $\boldsymbol{s}_1 \leftarrow$ `Support` $\left(\boldsymbol{A}_1\bar{\mathcal{B}}_1 \oplus \{\boldsymbol{v}_1\}, -\boldsymbol{c}\right)$
    $\boldsymbol{s}_2 \leftarrow$ `Support` $\left(\boldsymbol{A}_2\bar{\mathcal{B}}_2 \oplus \{\boldsymbol{v}_2\}, \boldsymbol{c}\right)$
    $V \leftarrow (\boldsymbol{s}_2 - \boldsymbol{s}_1) \cdot \boldsymbol{c}/d$ `// Eq. (22)`
    if $V \le 0$ **then**
      return 1.0
    else
      $\hat{t}^* \leftarrow d/V$
      $t_\alpha \leftarrow \beta\hat{t}^*$ `// Sec. 4.3`
    end
    $d, \boldsymbol{c} \leftarrow$ `GJK` $\left(\mathcal{B}_1(T + t_\alpha), \mathcal{B}_2(T + t_\alpha)\right)$
    if $T > 0$ **and** $d \le s\bar{d}$ **then**
      return T
    end
    $T \leftarrow T + t_\alpha$
    if $T \ge 1.0$ **then**
      return 1.0
    end
    $\beta \leftarrow 1 - \frac{1}{\alpha}$
  end

5 APPROXIMATED RELATIVE MOTION

In Section 4, we discussed the CCD between two moving shapes, where we made two independent relaxations in Equation (11). The number of relaxations can be reduced to one when only one shape moves while the other is static, which leads to a tighter estimate $\hat{t}^*$ and faster convergence. When handling two moving shapes, it is natural to consider the motion of $\mathcal{B}_2$ *relative* to $\mathcal{B}_1$, i.e., $\mathcal{B}_{2[1]} = \mathcal{B}_{2[1]}(t)$, as observed from the frame attached to $\mathcal{B}_1$. Unfortunately, as we will show later, $\mathcal{B}_{2[1]}$ does not undergo a *constant* affine motion. However, this issue can be addressed through approximation.

Section 5.1 presents a constant affine approximation of this relative motion so that the techniques discussed in Section 4 can be applied. Furthermore, Section 5.2 shows another merit of this approximation that allows *point-wise* computation for a tighter bound on t^*.

5.1 C^{5}D-Quad

Let Frame 0 denote the world frame, the observation frame used in Section 4. Let Frame 1 and 2 denote the frames attached to $\mathcal{B}_1, \mathcal{B}_2$, respectively. At time $t = 0$, the three frames coincide, and thus $\bar{\mathcal{B}}_{i[0]} = \bar{\mathcal{B}}_{i[1]} = \bar{\mathcal{B}}_{i[2]}$, all of which are denoted by $\bar{\mathcal{B}}_i$ for simplicity. Over time t, Frames 1 and 2 undergo constant affine motion observed from Frame 0:

$$\boldsymbol{T}_{i[0]}(t) = \begin{pmatrix} \boldsymbol{I} + t\boldsymbol{A}_{i[0]} & t\boldsymbol{v}_{i[0]} \\ \boldsymbol{0}^\top & 1 \end{pmatrix}, \quad i = 1, 2. \tag{24}$$

Note that each shape stays static when observed from its own frame, i.e., $\mathcal{B}_{i[i]}(t) \equiv \mathcal{B}_{i[i]}(0) = \bar{\mathcal{B}}_i$, and we have $\mathcal{B}_{i[0]}(t) = \boldsymbol{T}_{i[0]}(t)\mathcal{B}_{i[i]}(t) = \boldsymbol{T}_{i[0]}(t)\bar{\mathcal{B}}_i$ as in Equation (8). The "relative" transformation from Frame 1 to Frame 2 is:

$$\boldsymbol{T}_{2[1]}(t) = \boldsymbol{T}_{0[1]}(t)\boldsymbol{T}_{2[0]}(t) = \boldsymbol{T}_{1[0]}^{-1}(t)\boldsymbol{T}_{2[0]}(t) \tag{25}$$

$$= \begin{pmatrix} (\boldsymbol{I} + t\boldsymbol{A}_{1[0]})^{-1}(\boldsymbol{I} + t\boldsymbol{A}_{2[0]}) & t(\boldsymbol{I} + t\boldsymbol{A}_{1[0]})^{-1}\boldsymbol{v}_{12[0]} \\ \boldsymbol{0}^\top & 1 \end{pmatrix}, \tag{26}$$

where $\boldsymbol{v}_{12[0]} = \boldsymbol{v}_{2[0]} - \boldsymbol{v}_{1[0]}$. Note that $\boldsymbol{T}_{2[1]}(t)$ is not linear in t and thus $\mathcal{B}_{2[1]}$ does not undergo a *constant* affine motion. We approximate it with its first-order Taylor expansion:

$$\hat{\boldsymbol{T}}_{2[1]}(t) = \begin{pmatrix} \boldsymbol{I} + t\boldsymbol{A}_{12[0]} & t\boldsymbol{v}_{12[0]} \\ \boldsymbol{0}^\top & 1 \end{pmatrix}, \tag{27}$$

where $\boldsymbol{A}_{12[0]} = \boldsymbol{A}_{2[0]} - \boldsymbol{A}_{1[0]}$. The error of this approximation is analyzed by Claim 1, whose proof is provided in the appendix (Section B).

Claim 1. $\forall K > 1, \forall 0 \le t \le (1 - 1/K)/\|\boldsymbol{A}_{1[0]}\|, \forall \boldsymbol{b} \in \mathbb{R}^3$,

$$\left\|\left(\boldsymbol{T}_{2[1]}(t) - \hat{\boldsymbol{T}}_{2[1]}(t)\right)\boldsymbol{b}\right\| \le t^2K(\alpha\|\boldsymbol{b}\| + \beta), \tag{28}$$

where $\alpha = \|\boldsymbol{A}_{1[0]}\|\left(\|\boldsymbol{A}_{1[0]}\| + \|\boldsymbol{A}_{2[0]}\|\right)$, $\beta = \|\boldsymbol{A}_{1[0]}\|\|\boldsymbol{v}_{12[0]}\|$, *all vector norms are* l_2 *norms, and all matrix norms are* l_2 *spectral norms.*

Let $K > 1$ be a user-specified hyper-parameter, e.g., $K = 1.1$. By Claim 1, at time t $(0 \le t \le (1 - 1/K)/\|\boldsymbol{A}_{1[0]}\|)$, $\forall \bar{\boldsymbol{b}}_2 \in \bar{\mathcal{B}}_2$,

$$\boldsymbol{b}_2(t) = \boldsymbol{T}_{2[1]}(t)\bar{\boldsymbol{b}}_2 \in \left\{\hat{\boldsymbol{T}}_{2[1]}(t)\bar{\boldsymbol{b}}_2\right\} \oplus t^2\mathcal{S}\left(K(\alpha\|\bar{\boldsymbol{b}}_2\| + \beta)\right), \tag{29}$$

where $\mathcal{S}(r) = \{\boldsymbol{x} \mid \|\boldsymbol{x}\| \le r\}$ is the sphere of radius r centered at $\boldsymbol{0}$. Therefore, let $R_2 = K(\alpha \max_{\bar{\boldsymbol{b}}_2 \in \bar{\mathcal{B}}_2} \|\bar{\boldsymbol{b}}_2\| + \beta)$, and we have

$$\mathcal{B}_{2[1]}(t) = \boldsymbol{T}_{2[1]}(t)\bar{\mathcal{B}}_2 \subseteq \hat{\boldsymbol{T}}_{2[1]}(t)\bar{\mathcal{B}}_2 \oplus t^2\mathcal{S}(R_2) \tag{30}$$

$$\subseteq \bar{\mathcal{B}}_2 \oplus t\boldsymbol{A}_{12}\bar{\mathcal{B}}_2 \oplus t\{\boldsymbol{v}_{12}\} \oplus t^2\mathcal{S}(R_2), \tag{31}$$

where the "[0]"'s are omitted. Unlike in Equation (11), here no relaxation is needed for body 1 since $\mathcal{B}_{1[1]}(t) \equiv \bar{\mathcal{B}}_1$. Together,

$$\bar{\mathcal{B}}_1 \ominus \mathcal{B}_{2[1]}(t) \subseteq \bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2 \ominus t\left(\boldsymbol{A}_{12}\bar{\mathcal{B}}_2 \oplus \{\boldsymbol{v}_{12}\}\right) \ominus t^2\mathcal{S}(R_2). \tag{32}$$

Similar to Equation (21),

$$d(\bar{\mathcal{B}}_1 \ominus \mathcal{B}_{2[1]}(t)) \ge d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) - tV - t^2R_2, \tag{33}$$

where $\hat{\boldsymbol{n}}^* = \boldsymbol{c}(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2)/d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2)$, $V = (\boldsymbol{s}_{\boldsymbol{A}_{12}\bar{\mathcal{B}}_2}(\hat{\boldsymbol{n}}^*) - \boldsymbol{v}_{12}) \cdot \hat{\boldsymbol{n}}^*$. When $R_2 > 0$, a conservative estimate $\hat{t}^*$ can be obtained by solving the quadratic equation in t: $d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) - tV - t^2R_2 = 0$, which has exactly one positive root $\hat{t}^*$ since $-R_2 < 0$ and $d(\bar{\mathcal{B}}_1 \ominus \bar{\mathcal{B}}_2) > 0$; when $R_2 = 0$, the problem reduces to what has been discussed in Section 4.2. Note that $\hat{t}^*$ should be clamped into $\hat{t}^* < (1-1/K)/\|\boldsymbol{A}_1\|$ as required by Claim 1. After obtaining $\hat{t}^*$, the CCD problem can be

advanced as discussed in Section 4.3 before carrying out the next CA iteration.

It is worth noting that, alternative to approximating $\boldsymbol{T}_{2[1]}$, we can also choose Frame 2 as the reference frame and approximate $\boldsymbol{T}_{1[2]}$ to compute $\hat{t}^*(\mathcal{B}_{1[2]}, \bar{\mathcal{B}}_2)$ as the conservative estimate. In one iteration, different choices of the reference frame can result in different estimations of t^*. Figure 3 illustrates an example where the ground truth $t^* = +\infty$. When observed from Frame 0 or 2, $\mathcal{B}_{1[2]} = \mathcal{B}_{1[0]}$ is moving, resulting in a positive V and a small $\hat{t}^*$. In contrast, when observed from Frame 1, $\mathcal{B}_{2[1]}$ remains static, leading to a drastically larger $\hat{t}^*$. Due to the difficulty in determining the better reference frame in general, our method alternates between the two reference frames every iteration to leverage both. This summarizes the "C^{5}D-Quad" algorithm, with its pseudocode provided in Algorithm 2. This variant differs from Algorithm 1 mainly in its calculation of the time update $\hat{t}^*$.

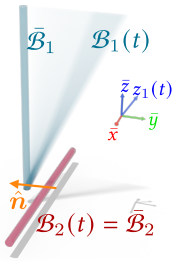

Fig. 3. A challenging case that simplifies when observed from Frame 2.

5.2 C^{5}D-Quad-Pw

In this section, we consider specifically the case where $\mathcal{B}_1$ and $\mathcal{B}_2$ are convex *polytopes*: $\mathcal{B}_i = \mathcal{H}(\mathcal{V}_i)$, where $\mathcal{V}_i \in \mathbb{R}^3$ is a finite set of vertices whose convex hull is denoted by $\mathcal{H}(\mathcal{V}_i) = \{\sum_j a_j \boldsymbol{v}_j \mid a_j \in \mathbb{R}, \boldsymbol{v}_j \in \mathcal{V}_i\}$, $i = 1, 2$. Let $\mathcal{V}_i(t) = \boldsymbol{T}_i \bar{\mathcal{V}}_i$.

Recall that for any unit vector $\hat{\boldsymbol{n}}$,

$$d(\mathcal{B}_1, \mathcal{B}_2) \geq \min_{\boldsymbol{b}_1 \in \mathcal{B}_1} \min_{\boldsymbol{b}_2 \in \mathcal{B}_2} (\boldsymbol{b}_1 - \boldsymbol{b}_2) \cdot \hat{\boldsymbol{n}} \tag{34}$$

$$= \min_{\boldsymbol{b}_1 \in \mathcal{B}_1} \boldsymbol{b}_1 \cdot \hat{\boldsymbol{n}} - \max_{\boldsymbol{b}_2 \in \mathcal{B}_2} \boldsymbol{b}_2 \cdot \hat{\boldsymbol{n}} = \min_{\boldsymbol{b}_1 \in \mathcal{V}_1} \boldsymbol{b}_1 \cdot \hat{\boldsymbol{n}} - \max_{\boldsymbol{b}_2 \in \mathcal{V}_2} \boldsymbol{b}_2 \cdot \hat{\boldsymbol{n}}, \tag{35}$$

which holds as equity when $\hat{\boldsymbol{n}} = \boldsymbol{c}(\mathcal{B}_1 \ominus \mathcal{B}_2)/d(\mathcal{B}_1 \ominus \mathcal{B}_2)$. Consider $\mathcal{B}_{1[1]}(t) \equiv \bar{\mathcal{B}}_1$ and $\mathcal{B}_{2[1]}(t)$. Let $P_{\hat{\boldsymbol{n}}} = \min_{\boldsymbol{b}_1 \in \bar{\mathcal{V}}_1} \boldsymbol{b}_1 \cdot \hat{\boldsymbol{n}}$. Then,

$$d\big(\bar{\mathcal{B}}_1, \mathcal{B}_{2[1]}(t)\big) \geq 0 \Leftarrow \forall \bar{\boldsymbol{b}} \in \bar{\mathcal{V}}_2, P_{\hat{\boldsymbol{n}}} - \boldsymbol{T}_{2[1]}(t)\bar{\boldsymbol{b}} \cdot \hat{\boldsymbol{n}} \geq 0 \tag{36}$$

$$\Leftarrow \forall \bar{\boldsymbol{b}} \in \bar{\mathcal{V}}_2, P_{\hat{\boldsymbol{n}}} - \hat{\boldsymbol{T}}_{2[1]}(t)\bar{\boldsymbol{b}} \cdot \hat{\boldsymbol{n}} - t^2 K(\alpha \|\bar{\boldsymbol{b}}\| + \beta) \geq 0, \tag{37}$$

where $\hat{\boldsymbol{T}}_{2[1]}(t)\bar{\boldsymbol{b}} = \bar{\boldsymbol{b}} + t(\boldsymbol{A}_{12[0]}\bar{\boldsymbol{b}} + \boldsymbol{v}_{12[0]})$. For every $\bar{\boldsymbol{b}} \in \bar{\mathcal{V}}_2$, A *point-wise* TOI $\hat{t}^*_{\bar{\boldsymbol{b}}}$ can be solved from the quadratic inequality in Equation (37), and we can take $\hat{t}^* = \min_{\bar{\boldsymbol{b}}} \hat{t}^*_{\bar{\boldsymbol{b}}}$ for a conservative estimate of the global TOI $t^*(\bar{\mathcal{B}}_1, \mathcal{B}_{2[1]})$. We refer to this method as the "C^{5}D-Quad-Pw" algorithm, with its pseudocode provided in Algorithm 3.

6 EVALUATION

This section evaluates our methods from two perspectives. First, we assess their performance on a synthetic dataset consisting of random pairs of convex polytopes undergoing constant affine motions, allowing for a general comparison with the baseline. Next, we integrate our CCD methods into the ABD framework to evaluate their practical performance. All experiments are conducted on an Intel Core i9-13900KS processor, with the implementation written in C++ and compiled using GCC with the -O2 optimization flag enabled.

Algorithm 2: C^{5}D-Quad

Procedure C^{5}D-Quad($\bar{\mathcal{B}}_1, \bar{\mathcal{B}}_2, \bar{\boldsymbol{A}}_1, \bar{\boldsymbol{A}}_2, \bar{\boldsymbol{v}}_1, \bar{\boldsymbol{v}}_2, \boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{v}_1, \boldsymbol{v}_2, s, \alpha, K$):
- **Input:** $\bar{\mathcal{B}}_i, \bar{\boldsymbol{A}}_i, \bar{\boldsymbol{v}}_i, \boldsymbol{A}_i, \boldsymbol{v}_i$: motion $\mathcal{B}_i(t) = (\boldsymbol{I} + t\boldsymbol{A}_i)\bar{\mathcal{B}}_i \oplus \{t\boldsymbol{v}_i\}$
  s, α: scaling factors required by conservative CCD
  K: the hyperparameter (Section 5.1)
- **Output:** $\hat{T}^*$: The conservative TOI
- $a, b \leftarrow 1, 2$
- $d, \boldsymbol{c} \leftarrow$ GJK $\big(\mathcal{B}_a(0), \mathcal{B}_b(0)\big)$
- $\bar{d} \leftarrow d$
- $T \leftarrow 0$
- $\beta \leftarrow 1 - s$
- **while** *true* **do**
  - /* Rewrite the motion starting from T using Eq. (7) */
  - $\boldsymbol{A}'_a,\ \boldsymbol{A}'_b \leftarrow \boldsymbol{A}_a(\bar{\boldsymbol{A}}_a + T\boldsymbol{A}_a)^{-1},\ \boldsymbol{A}_b(\bar{\boldsymbol{A}}_b + T\boldsymbol{A}_b)^{-1}$
  - $\boldsymbol{v}'_a,\ \boldsymbol{v}'_b \leftarrow \boldsymbol{v}_a - \boldsymbol{A}'_a(\bar{\boldsymbol{v}}_a + T\boldsymbol{v}_a),\ \boldsymbol{v}_b - \boldsymbol{A}'_b(\bar{\boldsymbol{v}}_b + T\boldsymbol{v}_b)$
  - $\boldsymbol{s}_{ab} \leftarrow$ Support $\big((\boldsymbol{A}'_b - \boldsymbol{A}'_a)\mathcal{B}_b(T) \oplus \{\boldsymbol{v}'_b - \boldsymbol{v}'_a\}, \boldsymbol{c}\big)$
  - $V \leftarrow \boldsymbol{s}_{ab} \cdot \boldsymbol{c}/d$
  - $r_b \leftarrow \max_{\boldsymbol{b} \in \mathcal{B}_b(T)} \|\boldsymbol{b}\|$
  - $R_b \leftarrow K\|\boldsymbol{A}'_a\|\big((\|\boldsymbol{A}'_a\| + \|\boldsymbol{A}'_b\|)r_b + \|\boldsymbol{v}'_b - \boldsymbol{v}'_a\|\big)$
  - $\Delta \leftarrow V^2 + 4dR_b$ // Solve Eq. (33)
  - $\hat{t}^* \leftarrow \min\big((V + \sqrt{\Delta})/(2R_b), (1 - 1/K)\|\boldsymbol{A}'_a\|\big)$
  - $t_\alpha \leftarrow \beta\hat{t}^*$
  - Swap(a, b)
  - $d, \boldsymbol{c} \leftarrow$ GJK $\big(\mathcal{B}_a(T + t_\alpha), \mathcal{B}_b(T + t_\alpha)\big)$
  - **if** $T > 0$ **and** $d \leq s\bar{d}$ **then**
    - **return** T
  - **end**
  - $T \leftarrow T + t_\alpha$
  - **if** $T \geq 1.0$ **then**
    - **return** 1.0
  - **end**
  - $\beta \leftarrow 1 - \frac{1}{\alpha}$
- **end**

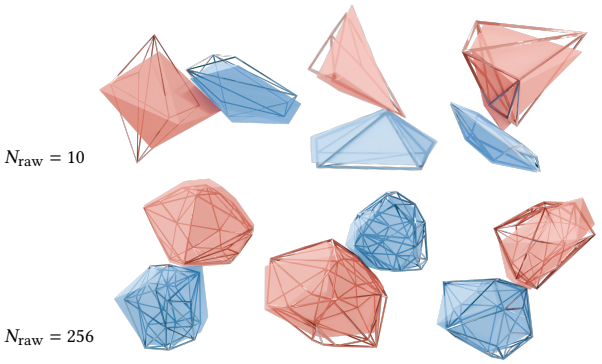

Fig. 4. **Synthetic data**: Example data points from the synthetic dataset (see Section 6.1). The wireframes represent the initial shapes, while the solid, semi-transparent objects indicate the transformed shapes at the ground-truth TOI.

Algorithm 3: C^{5}D-Quad-Pw

Procedure
C^{5}D-Quad-Pw($\bar{\mathcal{V}}_1, \bar{\mathcal{V}}_2, \bar{\boldsymbol{A}}_1, \bar{\boldsymbol{A}}_2, \bar{\boldsymbol{v}}_1, \bar{\boldsymbol{v}}_2, \boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{v}_1, \boldsymbol{v}_2, s, \alpha, K$):
Input: $\bar{\mathcal{V}}_i$: vertices of $\bar{\mathcal{B}}_i = \mathcal{H}(\bar{\mathcal{V}}_i)$
$\bar{\boldsymbol{A}}_i, \bar{\boldsymbol{v}}_i, \boldsymbol{A}_i, \boldsymbol{v}_i$: motion $\mathcal{B}_i(t) = (\boldsymbol{I} + t\boldsymbol{A}_i)\bar{\mathcal{B}}_i \oplus \{t\boldsymbol{v}_i\}$
s, α: scaling factors required by conservative CCD
K: the hyperparameter (Section 5.1)
Output: $\hat{T}^*$: The conservative TOI
$a, b \leftarrow 1, 2$
$d, \boldsymbol{c} \leftarrow$ GJK $\left(\mathcal{B}_a(0), \mathcal{B}_b(0)\right)$
$\bar{d} \leftarrow d$
$T \leftarrow 0$
$\beta \leftarrow 1 - s$
while *true* **do**
$\boldsymbol{A}'_a,\ \boldsymbol{A}'_b \leftarrow \boldsymbol{A}_a(\bar{\boldsymbol{A}}_a + T\boldsymbol{A}_a)^{-1},\ \boldsymbol{A}_b(\bar{\boldsymbol{A}}_b + T\boldsymbol{A}_b)^{-1}$
$\boldsymbol{v}'_a,\ \boldsymbol{v}'_b \leftarrow \boldsymbol{v}_a - \boldsymbol{A}'_a(\bar{\boldsymbol{v}}_a + T\boldsymbol{v}_a),\ \boldsymbol{v}_b - \boldsymbol{A}'_b(\bar{\boldsymbol{v}}_b + T\boldsymbol{v}_b)$
$\hat{\boldsymbol{n}} \leftarrow \boldsymbol{c}/d$
$P \leftarrow \hat{\boldsymbol{n}} \cdot$ Support $(\mathcal{B}_a(T), -\hat{\boldsymbol{n}})$
$\hat{t}^* \leftarrow (1 - 1/K)\|\boldsymbol{A}'_a\|$
for $\bar{\boldsymbol{b}} \in \bar{\mathcal{V}}_b$ **do**
$\boldsymbol{b}_T \leftarrow (\bar{\boldsymbol{A}}_b + t\boldsymbol{A}_b)\bar{\boldsymbol{b}} + \bar{\boldsymbol{v}}_b + t\boldsymbol{v}_b$
$A \leftarrow -K\|\boldsymbol{A}'_a\|\left(\left(\|\boldsymbol{A}'_a\| + \|\boldsymbol{A}'_b\|\right)\|\boldsymbol{b}_T\| + \|\boldsymbol{v}'_b - \boldsymbol{v}'_a\|\right)$
$B \leftarrow -\hat{\boldsymbol{n}} \cdot \left((\boldsymbol{A}'_b - \boldsymbol{A}'_a)\boldsymbol{b}_T + \boldsymbol{v}'_b - \boldsymbol{v}'_a\right)$
$C \leftarrow P - \boldsymbol{b}_T \cdot \hat{\boldsymbol{n}}$
$\Delta \leftarrow B^2 - 4AC$ // Solve Eq. (37)
$\hat{t}^* \leftarrow \min\left((-B + \sqrt{\Delta})/(2A), \hat{t}^*\right)$
end
$t_\alpha \leftarrow \beta\hat{t}^*$
Swap(a, b)
$d, \boldsymbol{c} \leftarrow$ GJK $\left(\mathcal{B}_a(T + t_\alpha), \mathcal{B}_b(T + t_\alpha)\right)$
if $T > 0$ **and** $d \le s\bar{d}$ **then**
return T
end
$T \leftarrow T + t_\alpha$
if $T \ge 1.0$ **then**
return 1.0
end
$\beta \leftarrow 1 - \frac{1}{\alpha}$
end

6.1 Synthetic Data

The synthetic dataset contains randomly sampled pairs of convex shapes and their constant affine motion. The convex shapes $\bar{\mathcal{B}}_1, \bar{\mathcal{B}}_2$ are generated by taking the convex hulls of N_{raw} points sampled from the standard normal distribution. We generate two subsets of data with $N_{\text{raw}} = 10$ and $N_{\text{raw}} = 256$ respectively to evaluate the CCD methods on both low- and high-resolution meshes. The initial transformation matrices $\bar{\boldsymbol{A}}_0, \bar{\boldsymbol{A}}_1$ are generated by adding a small Gaussian noise to randomly sampled rotation matrices, mimicking the common case in ABD simulation in practice, where the stiff objects simulated always have near-rigid affine transformations. Other information about the constant affine motion, including $\bar{\boldsymbol{v}}_0, \bar{\boldsymbol{v}}_1, \boldsymbol{A}_0, \boldsymbol{A}_1, \boldsymbol{v}_0, \boldsymbol{v}_1$, are also randomly sampled. More details about the sampling process and the random distributions can be found in the appendix (Section F).

Table 1. The evaluation results on synthetic data. Numbers with standard deviation (std) are formatted as "mean ± std". N_{cvx}: mean number of vertices on the convex collision shapes, i.e. the convex hulls of the N_{raw} random points. (*): The largest number of iterations among all primitive pairs of each data sample.

N_{raw} (N_{cvx})	Method	Iter	Key Iter (P_k = 0.9, 0.95, 0.99)	Time (μs)
10 (8.2)	ACCD	76.4 ± 93.8*	–	60 ± 16
	C^{5}D-Linear	32.9 ± 40.9	6.2, 7.7, 11.3	**20** ± 26
	C^{5}D-Quad	27.5 ± 29.3	6.5, 7.9, 10.8	42 ± 45
	C^{5}D-Quad-Pw	**16.3** ± 10.6	**4.6, 5.4, 6.9**	25 ± 16
256 (28.2)	ACCD	103.9 ± 119.5*	–	719 ± 141
	C^{5}D-Linear	40.9 ± 55.3	7.9, 9.8, 14.2	48 ± 60
	C^{5}D-Quad	34.2 ± 37.0	7.7, 9.3, 12.9	73 ± 75
	C^{5}D-Quad-Pw	**14.3** ± 7.9	**3.9, 4.6, 5.9**	**32** ± 16

The "ground-truth" ToI in our synthetic dataset is obtained by running ACCD [Li et al. 2021] until convergence as defined in Equation 2 with $s = 10^{-6}$. We reject data points whose shapes initially collide ($\hat{d} = 0$) or whose ToI is smaller than 0.01 or larger than 1.0. Figure 4 presents several examples from the synthetic dataset.

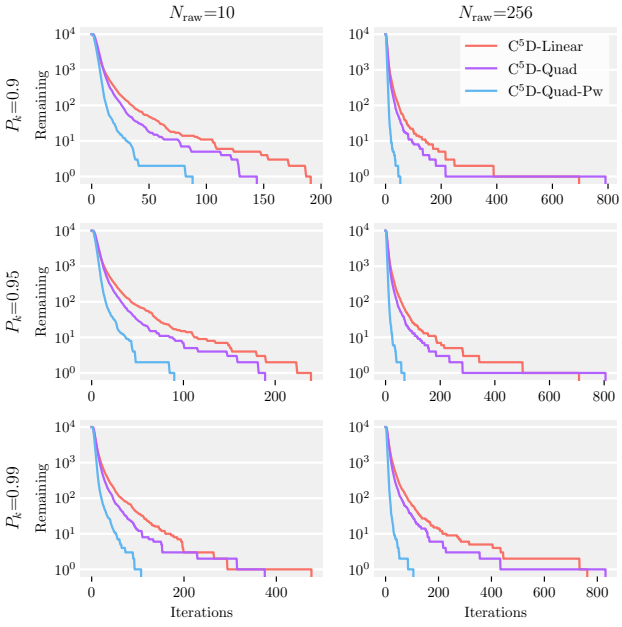

Fig. 5. **Convergence curve**: Log-scale convergence curves on synthetic data. X-axis: number of CA iterations. Y-axis: number of remaining data samples with $\hat{t}^* < P_k t^*$. C^{5}D-Quad-Pw demonstrates efficient and stable convergence, even on data samples that pose challenges for C^{5}D-Linear and C^{5}D-Quad.

Metrics. The evaluation metrics include the runtime and the number of CA iterations taken before convergence. For ACCD, every primitive pair takes a potentially different number of iterations, and

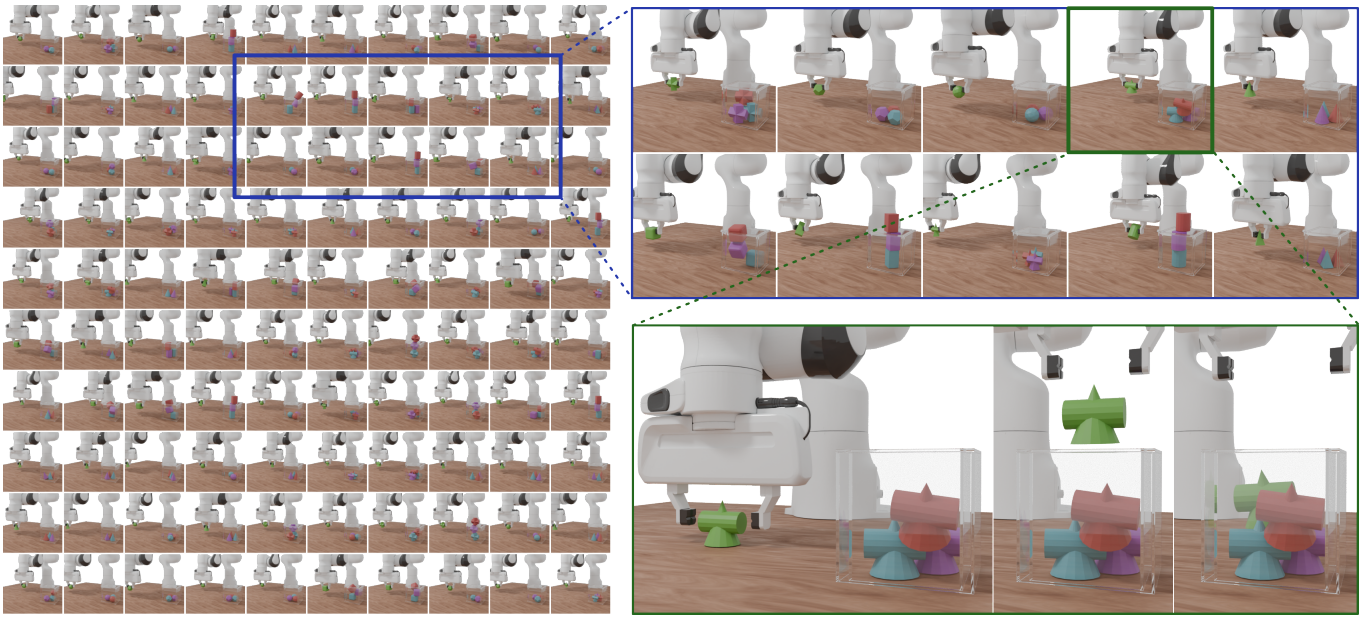

Fig. 6. **Pick**: A Franka Hand picks up objects and drops them into a box, with the finger trajectories generated using motion planning. Left image: A "batch" of 100 similar scenes with different initializations. When simulated sequentially on a single thread, our C^{5}D-Linear method achieves a 10× speed-up over the ACCD baseline. Scene-level parallelism on a 24-core CPU further boosts throughput by approximately 14× for both methods.

we present only the worst-case (i.e., largest) number among all primitive pairs. Additionally, to assess the convergence rate of different variants of C^{5}D, we also evaluate the numbers of iterations taken before the newest lower-bound estimation of ToI, $\hat{t}^*$, reaches certain "key proportions" P_k of the ground truth ToI t^*. Our experiment results present the corresponding "key number of iterations" for $P_\text{k} = 0.9, 0.95$, and 0.99. The corresponding convergence curves are visualized in Figure 5.

We evaluate three variants of our method (C^{5}D-Linear, C^{5}D-Quad, and C^{5}D-Quad-Pw) against the ACCD baseline [Li et al. 2021]. Table 1 presents experimental results on synthetic data, using 10,000 samples for each value of N_raw.

Discussion. On both subsets with low- and high-resolution meshes, C^{5}D methods outperform the ACCD baseline in terms of runtime, benefitting from avoiding traversing the primitive pairs. Moreover, the mean number of iterations of C^{5}D is also notably smaller than the mean worst-case number of iterations of ACCD. This observation indicates that, although most primitive pairs are easy, some primitive pairs are especially challenging for ACCD, bringing down its overall performance.

Among all the C^{5}D variants, C^{5}D-Quad-Pw requires the fewest CA iterations on both subsets. However, for the $N_\text{raw} = 10$ subset, C^{5}D-Linear achieves faster runtime despite requiring more iterations. This is likely due to the simplicity of the iterations in C^{5}D-Linear, which are more easily optimized by the compiler. Nevertheless, Figure 5 shows that on some rare but challenging data samples, C^{5}D-Linear requires significantly more iterations to converge, whereas C^{5}D-Quad-Pw maintains a more stable iteration count, attributed to its robustness as discussed in Section 5.1.

6.2 Application to ABD

To evaluate the performance of our C^{5}D methods in practice, we integrate them into our implementation of ABD and assess the efficiency of different CCD methods in the simulation of a series of test scenes. We use ACCD [Li et al. 2021] as the baseline and compare it against our proposed C^{5}D-Linear and C^{5}D-Quad-Pw methods. The C^{5}D-Quad variant is excluded from this comparison for simplicity, as it is consistently outperformed by C^{5}D-Quad-Pw. The test scenes and corresponding CCD performances are summarized in Table 2. The multi-threaded functions are implemented using OpenMP. Detailed information regarding the experimental setup and the metrics is provided in the remainder of this section.

Test scenes. In the test scenes, all collision shapes are represented by triangular meshes. Some scenes, e.g. "Bunnies" and "Gears", contain originally non-convex bodies. The collision shapes of those bodies are represented by unions of multiple convex meshes, either manually designed or generated using CoACD [Wei et al. 2022]. This collision shape representation is naturally supported by both ACCD and C^{5}D in ABD simulation. For a fair comparison, we use the same collision shapes for all CCD methods, although ACCD does not require the convexity of collision shapes.

We test the multi-threaded performance of the CCD methods on the "Pick" scene (Figure 6). This scene aims to simulate the application scenario of training a robot arm for a pick-and-place task. We sample a "batch" of initial configurations with random object shapes (selected from a set of 10 shapes) and initial positions, and use the motion planning library MPlib [Guo et al. 2021] to generate the pick-and-place trajectories for each initialization. The trajectories of the two fingers (each with one convex collision shape)

Fig. 7. **Dummies**: This scene involves 60 dummies falling onto the ground, each composed of 15 convex links connected by ball joints. In this scene, our C^{5}D-Linear method achieves a 17× speed-up compared to the ACCD baseline.

Table 2. Evaluating the CCD performance during ABD simulation. **Bdy**: the number of dynamic affine bodies in the scene, excluding static obstacles. **Vtx**: the total number of mesh vertices of all bodies. **Cvx**: the total number of convex shapes in the convex decomposition of all bodies. **Queries**: the mean number of CCD queries per time step. **Iter**: the mean total number of CA iterations taken by all CCD queries before convergence in a time step. **Max Iter**: the mean maximum number of CA iterations among all CCD queries in a time step. **Time**: the mean CCD runtime per time step in the ABD simulation. **(†)**: parallelized implementation running on a 24-core CPU.

Scene (Fig.) [Bdy, Vtx, Cvx]	Method	Queries	Iter	Max Iter	Time (ms)
Dummies (7) [900, 45K, 901]	ACCD	2.6×10^7	2.6×10^7	261.5	9461.2
	C^{5}D-Linear	9.4×10^4	$\mathbf{9.5 \times 10^4}$	**28.4**	**547.9**
	C^{5}D-Quad-Pw	9.4×10^4	1.9×10^5	38.2	1094.4
Bunnies (8) [54, 64K, 275]	ACCD	7.0×10^6	7.1×10^6	81.8	1129.2
	C^{5}D-Linear	7.3×10^2	$\mathbf{1.3 \times 10^4}$	64.2	**210.6**
	C^{5}D-Quad-Pw	7.3×10^2	1.5×10^4	**13.4**	265.3
Gears (10) [14, 11K, 396]	ACCD	4.4×10^5	4.5×10^5	18.5	71.9
	C^{5}D-Linear	1.0×10^3	$\mathbf{1.2 \times 10^3}$	16.8	**5.3**
	C^{5}D-Quad-Pw	1.0×10^3	2.2×10^3	**8.7**	9.8
Gachapon (1) [11, 3K, 129]	ACCD	1.3×10^6	1.3×10^6	15.9	209.6
	C^{5}D-Linear	4.4×10^3	$\mathbf{4.4 \times 10^3}$	8.6	**16.8**
	C^{5}D-Quad-Pw	4.4×10^3	8.8×10^3	**2.6**	32.4
Wrecking Ball (9) [518, 5K, 519]	ACCD	1.8×10^6	1.8×10^6	15.9	291.5
	C^{5}D-Linear	2.9×10^4	$\mathbf{2.9 \times 10^4}$	**3.5**	**44.3**
	C^{5}D-Quad-Pw	2.9×10^4	5.8×10^4	4.2	114.9
Pick (6) [6, 328.0, 12.2]	ACCD	3.5×10^4	3.5×10^4	1.8	5.81
	C^{5}D-Linear	2.2×10^2	$\mathbf{2.2 \times 10^2}$	**1.4**	**0.57**
	C^{5}D-Quad-Pw	2.2×10^2	4.3×10^2	2.2	1.23
	ACCD-prim†	3.5×10^4	3.5×10^4	1.8	2.28
	ACCD-scene†	3.5×10^4	3.5×10^4	1.8	0.40
	C^{5}D-Linear†	2.2×10^2	$\mathbf{2.2 \times 10^2}$	**1.4**	**0.043**
	C^{5}D-Quad-Pw†	2.2×10^2	4.3×10^2	2.2	0.092

are passed to the ABD simulation as boundary conditions, whereas the other parts of the robot arm are intentionally designed by motion planning to avoid contact with other objects in the scene and are used solely for rendering purposes.

Metrics. We evaluate the CCD performances by measuring the total and maximum CA iterations and the total CCD runtime per time step. For each scene, we select 20 consecutive representative time steps that involve the highest number of inter-body collisions. All metrics are averaged over these selected time steps. To compare efficiency, we report the total number of iterations and total CCD runtime per time step. Additionally, to illustrate worst-case performance scenarios, we report the maximum number of iterations taken by CCD queries within each time step, also averaged over all time steps. For reference, we also provide the total number of CCD queries per time step, which corresponds to the number of active convex shape pairs for the C^{5}D methods or the number of active vertex-triangle and edge-edge pairs for ACCD. A collision pair is considered active if both colliding shapes reside within the i-AABB [Lan et al. 2022] of the two bodies.

For the multi-threaded experiment on the "Pick" scene, we employ two different forms of parallelism for ACCD: primitive-level (denoted by "ACCD-prim"), aimed at reducing latency, and scene-level (denoted by "ACCD-scene"), focusing on increasing throughput. Following prior works [Lan et al. 2022; Li et al. 2021], primitive-level parallelism assigns one thread to process each primitive pair in the scene once every sub-step, while different scene instances are simulated sequentially. In contrast, scene-level parallelism assigns one thread to simulate each scene instance, involving no inter-thread communication until the simulation of all scenes is completed. For C^{5}D methods, we only apply scene-level parallelism. For scene-level parallelism, the reported CCD runtime is a "wall time" estimated by scaling the overall simulation wall time by the proportion of CCD CPU runtime relative to the overall simulation CPU runtime.

Discussion. Overall, C^{5}D-Linear demonstrates the best efficiency among the three compared methods in terms of total runtime, being approximately 10 times faster than ACCD and 2 times faster than C^{5}D-Quad-Pw. A noteworthy observation is that the shape motions within one substep in actual ABD simulation are often very *simple* compared to those in the synthetic data in Section 6.1. These simple motions often have small affine velocity components, making them nearly translational or even static. In most test scenes, each CCD query takes barely more than one CA iteration for ACCD and C^{5}D-Linear. In these cases, C^{5}D-Quad-Pw suffers from the overly conservative bound on the affine velocity component, often requiring two iterations and resulting in being twice as slow as C^{5}D-Linear. However, in some rare but complex queries, C^{5}D-Quad-Pw can require fewer worst-case iterations, though it still falls behind in overall runtime.

On the other hand, ACCD is highly efficient in handling each individual query involving one pair of primitives, even though some complex queries might require more iterations. However, its overall

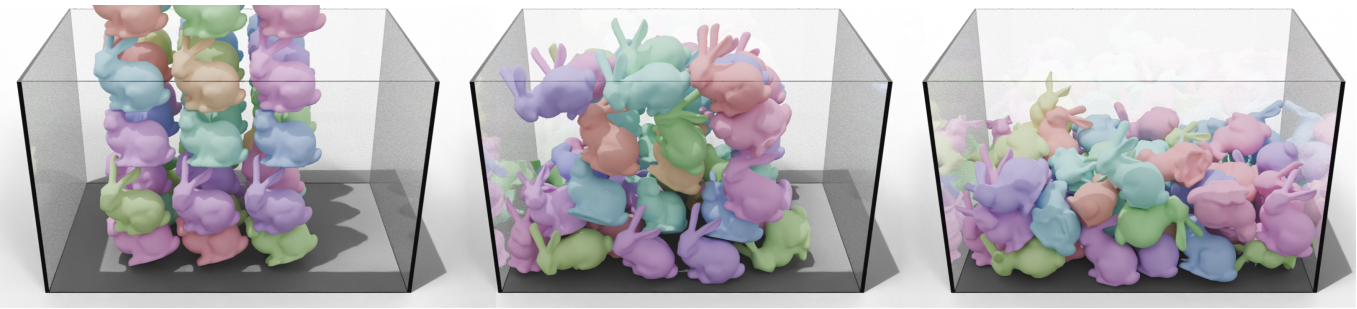

Fig. 8. **Bunnies**: We simulate 54 Stanford Bunnies falling into a glass box. The collision shape of each bunny is decomposed into 5 convex components using CoACD. Our C^{5}D-Linear method delivers a 5× speed-up over the ACCD baseline in this scene.

Fig. 9. **Wrecking ball**: A wrecking ball hits 512 cubes. Despite the simplicity of most collision shapes (each cube having only 8 vertices), our C^{5}D-Linear method achieves a 5× speed-up compared to the ACCD baseline.

performance is significantly constrained by the large number of queries generated by the numerous primitive pairs.

Primitive-level parallelization helps ACCD mitigate the challenge of processing numerous primitive pairs, achieving a 2.5-fold runtime reduction. However, the frequent communication overhead limits the effectiveness of this improvement, particularly when compared to scene-level parallelization which achieves a 14.5-fold speed-up. Scene-level parallelization also provides a comparable level of acceleration for the C^{5}D methods. As a result, C^{5}D-Linear remains the most efficient method among the multi-threaded implementations.

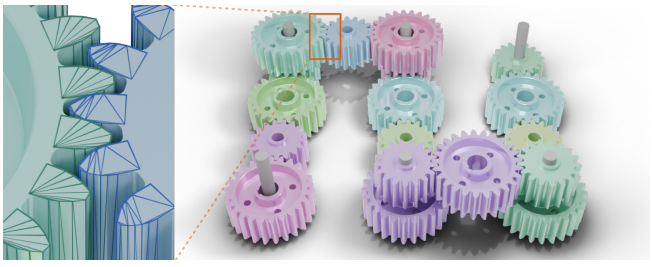

Fig. 10. **Gears**: A chain of 14 gears driven by the driver gear on the lower-left corner. Each tooth of a gear's collision shape is modeled convex. Our C^{5}D-Linear method delivers a 14× speed-up over the ACCD baseline in this scene.

7 LIMITATIONS AND FUTURE WORK

Our C^{5}D methods rely on two key assumptions about the collision shapes: convexity and affine motion. The convexity assumption restricts the applicability of our approach to highly non-convex objects, such as tubes and tori, whose convex decomposition may require an impractically large number of convex components to achieve a close-to-zero approximation error. On the other hand, the affine motion assumption limits the method's applicability to collision detection between highly deformable objects, such as cloth and ropes. While C^{5}D is able to handle the coupling between affine bodies and deformable objects by treating each primitive on the deformable object as a convex component, when both objects are deformable, it reduces to a primitive-level CCD method similar to ACCD and loses its efficiency advantages.

Looking ahead, we see several opportunities for further improving the efficiency of C^{5}D. For high-resolution convex meshes, one promising direction is to accelerate support mapping computations using spatial data structures [Guibas et al. 2000; Teschner et al. 2003; Wang et al. 2018]. Another potential enhancement is optimizing the usage of GJK by incorporating early-stopping criteria and reusing the simplex across CA iterations, inspired by [van den Bergen and Gregorius 2010]. These enhancements could make C^{5}D even more efficient, benefiting the realistic simulation of (nearly-)rigid objects and advancing its applications in graphics and robotics.

ACKNOWLEDGMENTS

We thank all the reviewers for their helpful comments and suggestions. This work was supported by Hillbot Inc. and NSF grant 2301040.

REFERENCES

Mihai Anitescu and Florian A Potra. 1997. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics* 14 (1997), 231–247.

David Baraff. 1994. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 23–34. https://doi.org/10.1145/192161.192168

David Baraff. 1995. Interactive Simulation of Solid Rigid Bodies. *IEEE Comput. Graph. Appl.* 15, 3 (May 1995), 63–75. https://doi.org/10.1109/38.376615

David Baraff. 1997. An introduction to physically based modeling: rigid body simulation I—unconstrained rigid body dynamics. *SIGGRAPH course notes* 82 (1997).

Klaus-Jürgen Bathe. 2006. *Finite element procedures*. Klaus-Jurgen Bathe.

Jan Bender, Kenny Erleben, and Jeff Trinkle. 2014. Interactive simulation of rigid body dynamics in computer graphics. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 246–270.

John Canny. 1986. Collision Detection for Moving Polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8, 2 (1986), 200–209. https://doi.org/10.1109/TPAMI.1986.4767773

Yunuo Chen, Minchen Li, Lei Lan, Hao Su, Yin Yang, and Chenfanfu Jiang. 2022. A unified newton barrier method for multibody dynamics. *ACM Trans. Graph.* 41, 4, Article 66 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530076

Erwin Coumans and Yunfei Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Evan Drumwright. 2007. A fast and stable penalty method for rigid body simulation. *IEEE transactions on visualization and computer graphics* 14, 1 (2007), 231–240.

Kenny Erleben. 2007. Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics (TOG)* 26, 2 (2007), 12–es.

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M Kaufman, and Daniele Panozzo. 2021. Intersection-free rigid body dynamics. *ACM Transactions on Graphics* 40, 4 (2021).

Susan Fisher and Ming C Lin. 2001. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, Vol. 1. IEEE, 330–336.

E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation* 4, 2 (1988), 193–203. https://doi.org/10.1109/56.2083

Leonidas J Guibas, David Hsu, and Li Zhang. 2000. A hierarchical method for real-time distance computation among moving convex bodies. *Computational Geometry* 15, 1-3 (2000), 51–68.

Runlin (Kolin) Guo, Xinsong Lin, Minghua Liu, Jiayuan Gu, and Hao Su. 2021. *MPlib: a Lightweight Motion Planning Library*. https://github.com/haosulab/MPlib

Shoichi Hasegawa, Nobuaki Fujii, Katsuhito Akahane, Yasuharu Koike, and Makoto Sato. 2004. Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Transactions of the Society of Instrument and Control Engineers* 40, 2 (2004), 122–131.

Kemeng Huang, Xinyu Lu, Huancheng Lin, Taku Komura, and Minchen Li. 2024. Advancing GPU IPC for stiff affine-deformable simulation. arXiv:2411.06224 [cs.GR] https://arxiv.org/abs/2411.06224

Daniel W. Johnson. 1987. *The optimization of robot motion in the presence of obstacles*. Ph. D. Dissertation. USA. AAI8720285.

Danny M Kaufman, Timothy Edmunds, and Dinesh K Pai. 2005. Fast frictional dynamics for rigid bodies. In *ACM SIGGRAPH 2005 Papers*. 946–956.

Lei Lan, Danny M. Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. 2022. Affine body dynamics: fast, stable and intersection-free simulation of stiff materials. *ACM Trans. Graph.* 41, 4, Article 67 (jul 2022), 14 pages. https://doi.org/10.1145/3528223.3530064

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4, Article 49 (2020).

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 170 (2021).

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*. 49–54.

Brian Mirtich and John Canny. 1995. Impulse-based simulation of rigid bodies. In *Proceedings of the 1995 symposium on Interactive 3D graphics*. 181–ff.

Brian Vincent Mirtich. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. Ph. D. Dissertation. University of California Berkeley. https://people.eecs.berkeley.edu/~jfc/mirtich/thesis/mirtichThesis.pdf

Mattia Montanari, Nik Petrinic, and Ettore Barbieri. 2017. Improving the GJK Algorithm for Faster and More Reliable Distance Queries Between Convex Objects. *ACM Trans. Graph.* 36, 3, Article 30 (June 2017), 17 pages. https://doi.org/10.1145/3083724

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

Xavier Provot. 1997. Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation'97: Proceedings of the Eurographics Workshop in Budapest, Hungary, September 2–3, 1997*. Springer, 177–189.

S. Redon, A. Kheddar, and S. Coquillart. 2000. An algebraic solution to the problem of collision detection for rigid polyhedral objects. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, Vol. 4. 3733–3738 vol.4. https://doi.org/10.1109/ROBOT.2000.845313

David E Stewart. 2000. Rigid-body dynamics with friction and impact. *SIAM review* 42, 1 (2000), 3–39.

Min Tang, Young J Kim, and Dinesh Manocha. 2009. C2A: Controlled conservative advancement for continuous collision detection of polygonal models. In *2009 IEEE International Conference on Robotics and Automation*. IEEE, 849–854.

Min Tang, Ruofeng Tong, Zhendong Wang, and Dinesh Manocha. 2014. Fast and exact continuous collision detection with bernstein sign classification. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–8.

Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. 2003. Optimized spatial hashing for collision detection of deformable objects.. In *Vmv*, Vol. 3. 47–54.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

Jeffrey C Trinkle, JA Tzitzouris, and Jong-Shi Pang. 2001. Dynamic multi-rigid-body systems with concurrent distributed contacts. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 359, 1789 (2001), 2575–2593.

Gino van den Bergen. 2004. *Collison detection in interactive 3D environments*. Elsevier/Morgan Kaufman, Amsterdam ; Boston.

Gino van den Bergen and Dirk Gregorius. 2010. *Game physics pearls*. A.K. Peters, Natick, Mass. 99–123 pages.

Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2021. A Large-scale Benchmark and an Inclusion-based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5, Article 188 (Sept. 2021), 16 pages. https://doi.org/10.1145/3460775

Xinlei Wang, Min Tang, Dinesh Manocha, and Ruofeng Tong. 2018. Efficient BVH-based collision detection scheme with ordering and restructuring. In *Computer graphics forum*, Vol. 37. Wiley Online Library, 227–237.

Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. 2022. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–18.

Hongyi Xu, Yili Zhao, and Jernej Barbič. 2014. Implicit multibody penalty-baseddistributed contact. *IEEE transactions on visualization and computer graphics* 20, 9 (2014), 1266–1279.

A THEOREM NEEDED BY EQUATION (20)

Theorem 1. *Let $\mathcal{B} \subset \mathbb{R}^3$ be convex, $\boldsymbol{a} \in \mathcal{B}$ s.t. $\forall \boldsymbol{b} \in \mathcal{B}, \|\boldsymbol{b}\| \geq \|\boldsymbol{a}\|$. Then $\forall \boldsymbol{b} \in \mathcal{B}, \boldsymbol{a} \cdot (\boldsymbol{b} - \boldsymbol{a}) \geq 0$.*

Proof. Suppose for contradiction that $\exists \boldsymbol{c} \in \mathcal{B}$ s.t. $\boldsymbol{a} \cdot (\boldsymbol{c} - \boldsymbol{a}) < 0$. If $\boldsymbol{c} \cdot (\boldsymbol{c} - \boldsymbol{a}) \leq 0$, then $\boldsymbol{c} \cdot \boldsymbol{c} \leq \boldsymbol{c} \cdot \boldsymbol{a} < \boldsymbol{a} \cdot \boldsymbol{a}$, contradicting that $\|\boldsymbol{c} \cdot \boldsymbol{c}\| \geq \|\boldsymbol{a} \cdot \boldsymbol{a}\|$; if $\boldsymbol{c} \cdot (\boldsymbol{c} - \boldsymbol{a}) > 0$, let $k = -\boldsymbol{a} \cdot (\boldsymbol{c} - \boldsymbol{a}) / \|\boldsymbol{c} - \boldsymbol{a}\|^2$, then $0 < k < 1$, hence $\boldsymbol{a} + k(\boldsymbol{c} - \boldsymbol{a}) \in \mathcal{B}$. But $\|\boldsymbol{a} + k(\boldsymbol{c} - \boldsymbol{a})\|^2 = \|\boldsymbol{a}\|^2 - (\boldsymbol{a} \cdot (\boldsymbol{c} - \boldsymbol{a}))^2 / \|\boldsymbol{c} - \boldsymbol{a}\|^2 < \|\boldsymbol{a}\|^2$, which is contradictory. □

Intuitively, Theorem 1 states that: if $\boldsymbol{a}$ is the closest point to the origin on a convex shape, $-\boldsymbol{a}$ serves as a *normal* vector. Specifically, the entire convex shape lies on one side of the plane that is perpendicular to $\boldsymbol{a}$ and passes through $\boldsymbol{a}$, while the origin lies on the other side.

B PROOF FOR CLAIM 1

We analyze the error in the linear transformation block $\boldsymbol{E_A}$ and the translation block $\boldsymbol{e_v}$ separately, as in

$$T_{2[1]}(t) - \hat{T}_{2[1]}(t) = \begin{pmatrix} \boldsymbol{E_A}(t) & \boldsymbol{e_v}(t) \\ \boldsymbol{0}^\top & 1 \end{pmatrix}, \tag{38}$$

Omitting the "[0]"'s for simplicity, the linear transformation error is

$$\begin{aligned} \boldsymbol{E_A}(t) &= (\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}(\boldsymbol{I} + t\boldsymbol{A}_2) - (\boldsymbol{I} + t\boldsymbol{A}_2 - t\boldsymbol{A}_1) && (39)\\ &= (\boldsymbol{I} + t\boldsymbol{A}_1)^{-1} - \boldsymbol{I} + t\big((\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}\boldsymbol{A}_2 - \boldsymbol{A}_2 + \boldsymbol{A}_1\big) && (40)\\ &= \big(\boldsymbol{I} - t\boldsymbol{A}_1 + t^2\boldsymbol{A}_1^2(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}\big) - \boldsymbol{I} && (41)\\ &\quad + t\big((\boldsymbol{I} - t\boldsymbol{A}_1(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1})\boldsymbol{A}_2 - \boldsymbol{A}_2 + \boldsymbol{A}_1\big) && (42)\\ &= t^2\boldsymbol{A}_1\big(\boldsymbol{A}_1(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1} - (\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}\boldsymbol{A}_2\big). && (43) \end{aligned}$$

Since $t\|\boldsymbol{A}_1\| \le 1 - 1/K < 1$ by the assumption of the claim, we can use the Neumann series to bound

$$\|(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}\| = \Big\|\sum_{j=0}^{\infty}(-t\boldsymbol{A}_1)^j\Big\| \le \sum_{j=0}^{\infty}(t\|\boldsymbol{A}_1\|)^j = \frac{1}{1 - t\|\boldsymbol{A}_1\|} \le K. \tag{44}$$

Therefore,

$$\|\boldsymbol{E_A}(t)\| \le t^2\|\boldsymbol{A}_1\|(\|\boldsymbol{A}_1\| + \|\boldsymbol{A}_2\|)(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1} = t^2K\alpha. \tag{45}$$

As for the translation block,

$$\begin{aligned} \boldsymbol{e_v}(t) &= t\big((\boldsymbol{I} + t\boldsymbol{A}_1)^{-1} - \boldsymbol{I}\big)\boldsymbol{v}_{12} && (46)\\ &= -t^2\boldsymbol{A}_1(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}\boldsymbol{v}_{12}, && (47) \end{aligned}$$

and

$$\|\boldsymbol{e_v}(t)\| \le t^2\|\boldsymbol{A}_1\|\|(\boldsymbol{I} + t\boldsymbol{A}_1)^{-1}\|\|\boldsymbol{v}_{12}\| \le t^2K\beta. \tag{48}$$

Finally, combining the norm bounds on $\boldsymbol{E_A}$ and $\boldsymbol{e_v}$, we have

$$\begin{aligned} \|T_{2[1]}(t) - \hat{T}_{2[1]}(t)\boldsymbol{b}\| &= \|\boldsymbol{E_A}\boldsymbol{b} + \boldsymbol{e_v}\| \le \|\boldsymbol{E_A}\|\|\boldsymbol{b}\| + \|\boldsymbol{e_v}\| && (49)\\ &\le t^2K(\alpha\|\boldsymbol{b}\| + \beta). && (50) \end{aligned}$$

C IMPLEMENTATION DETAILS

We implemented the "Support" function (see Algorithm 1) as a plain for loop through all vertices of the convex mesh $\mathcal{B}$, assuming the number of vertices is always small. For high-resolution meshes, this process could be accelerated using many possible choices of spatial data structures, such as BVH [Wang et al. 2018], hash grids [Teschner et al. 2003], and also data structures that are specially designed for convex polytopes [Guibas et al. 2000].

We follow Montanari et al. [2017] for the implementation of the GJK algorithm, which also calls the "Support" function. In the pseudocode, we directly pass the transformed convex shapes into the "Support" and "GJK" functions for readability; in our implementation, we pass the untransformed convex shapes $\bar{\mathcal{B}}$ with the corresponding transformations instead. In this way, inside the implementation of the "Support" function, we can transform the query vector $\boldsymbol{v}$ efficiently instead of transforming every vertex of $\bar{\mathcal{B}}$, as mentioned in Section 4.2.

The numerical stability of our CA-based CCD method is closely tied to the numerical stability of the GJK algorithm. We follow the implementation of Montanari et al. [2017] [1] on the choices of floating-point tolerance parameters in the GJK algorithm. Specifically, we set $\varepsilon_{\text{rel}} = 10^4 \times \varepsilon_{\text{M}}$ and $\varepsilon_{\text{tol}} = 10^2 \times \varepsilon_{\text{M}}$, where ε_{M} is the machine epsilon, $\varepsilon_{\text{M}} \approx 2.22 \times 10^{-16}$ for double precision. We refer the reader to [Montanari et al. 2017] for a detailed discussion on these choices, as well as the numerical stability of Johnson's Algorithm [Johnson 1987], which is also important for the robustness of GJK.

D ADVANCEMENT FOR CONSERVATIVE CCD

In this section, we discuss the details of the advancement scheme for the conservative CCD problem with proof of correctness.

We first recap the definition of conservative CCD given in Equation (2). In the conservative CCD problem, instead of finding an exact TOI T^* such that $d(\mathcal{B}_1(T^*), \mathcal{B}_2(T^*)) = 0$, we seek a conservative TOI $\hat{T}^*$ that maintains a small gap between the shapes:

$$s\bar{d} \le d\big(\mathcal{B}_1(\hat{T}^*)), \mathcal{B}_2(\hat{T}^*)\big) < \alpha s\bar{d}, \tag{51}$$

where $\bar{d} = d\big(\mathcal{B}_1(0)), \mathcal{B}_2(0)\big)$. The left inequality enforces that the gap remains above the minimal separation requirement $s\bar{d}$, while the right inequality ensures the gap does not become too large.

All C^{5}D methods share the same CA framework. In the k-th CA iteration, we first rewrite each motion $\mathcal{B}(t)$ into a motion starting from the current TOI estimate $T^{(k)}$ by Equation (7):

$$\mathcal{B}^{(k)}(t) = \mathcal{B}\big(T^{(k)} + t\big) = (\boldsymbol{I} + t\boldsymbol{A}')\mathcal{B}(T) + t\boldsymbol{v}'. \tag{52}$$

For convenience, define the distance $d(t) = d\big(\mathcal{B}_1(t), \mathcal{B}_2(t)\big)$ as a function of t, and similarly $d^{(k)}(t) = d\big(T^{(k)}+t\big) = d\big(\mathcal{B}_1^{(k)}(t), \mathcal{B}_2^{(k)}(t)\big)$. In C^{5}D-Linear, C^{5}D-Quad, and C^{5}D-Quad-Pw, the distance function is approximated with different lower bounds $\hat{d}^{(k)} \le d^{(k)}$ respectively:

$$\begin{aligned} \hat{d}_{\text{L}}^{(k)}(t) &= d^{(k)}(0) - tV, && \text{Eq. (21)} && (53)\\ \hat{d}_{\text{Q}}^{(k)}(t) &= d^{(k)}(0) - tV - t^2R, && \text{Eq. (33)} && (54)\\ \hat{d}_{\text{QP}}^{(k)}(t) &= \min_{\bar{\boldsymbol{b}} \in \mathcal{B}_2^{(k)}(0)} \bar{d}_{\bar{\boldsymbol{b}}} - tV_{\bar{\boldsymbol{b}}} - t^2R_{\bar{\boldsymbol{b}}}. && \text{Eq. (37)} && (55) \end{aligned}$$

The constant coefficients have been simplified in the above equations; for full details, please refer to the corresponding equations in the main paper. Notably, in Equation (55), we have $\min_{\bar{\boldsymbol{b}} \in \mathcal{B}_2^{(k)}(0)} \bar{d}_{\bar{\boldsymbol{b}}} = d^{(k)}(0)$, which implies $\hat{d}^{(k)}(0) = d^{(k)}(0)$ across all lower-bound definitions. Another key observation is that all three lower bounds are *concave*. In specific, $\hat{d}_{\text{L}}^{(k)}(t)$ is linear, $\hat{d}_{\text{Q}}^{(k)}(t)$ is a concave quadratic function (since $R > 0$), and $\hat{d}_{\text{QP}}^{(k)}(t)$, as a point-wise minimum of multiple concave functions, is likewise concave.

As is mentioned in Section 4.3, in the advancement step, the current time $T^{(k)}$ is advanced to $T^{(k+1)} = T^{(k)} + t_\alpha$, where

$$t_\alpha = \big(1 - \frac{1}{\alpha}\big)\hat{t}^*, \quad \hat{d}^{(k)}(\hat{t}^*) = 0. \tag{56}$$

[1] https://github.com/MattiaMontanari/openGJK

At t_α, by the concavity of $\hat{d}^{(k)}$ for all variant methods,

$$\hat{d}^{(k)}(t_\alpha) \geq \frac{1}{\alpha}\hat{d}^{(k)}(0) + \left(1-\frac{1}{\alpha}\right)\hat{d}^{(k)}(\hat{t}^*) \tag{57}$$

$$= \frac{1}{\alpha}d^{(k)}(0) \tag{58}$$

$$= \frac{1}{\alpha}d(T). \tag{59}$$

Recall that we exit the loop and return $\hat{T}^* \leftarrow T$ when we find $d(T+t_\alpha) < s\bar{d}$ for the first time. In this case,

$$\frac{1}{\alpha}d(T) \leq \hat{d}^{(k)}(t_\alpha) \leq d^{(k)}(t_\alpha) < s\bar{d}, \tag{60}$$

and hence $d(T) < \alpha s\bar{d}$. Since we did not exit the loop when we advanced to T, we immediately have $d(T) \geq s\bar{d}$. Therefore, both inequalities are satisfied in Equation (51).

In practice, IPC and ABD simulation often uses large values of s, e.g. $s = 0.1$, to keep a safer gap between objects and prevent numerical issues. For $s > \frac{1}{\alpha}$, it is necessary to take a smaller step $t_\alpha^{(0)} = (1-s)\hat{t}^{*(0)}$ instead of $\left(1-\frac{1}{\alpha}\right)\hat{t}^{*(0)}$ in the first iteration (i.e. when $T=0$). The former step size guarantees that

$$d^{(0)}(t_\alpha) \geq sd^{(k)}(0) = s\bar{d} \tag{61}$$

by Equation (58), and prevents stopping in the first iteration.

E CONVERGENCE OF CONSERVATIVE ADVANCEMENT

In this section, we prove the convergence of our CA methods. Inspired by [van den Bergen 2004], the proof utilizes the continuity of the mapping $f(T^{(k)}) = T^{(k+1)}$, where $T^{(k)}$ denotes the lower-bound TOI estimate at the k-th CA iteration.

Each variant of C^{5}D defines a different mapping:

$$f_{\mathrm{L}}(T) = T + d(T)/V(T), \qquad \text{Eq. (21)}$$

$$f_{\mathrm{Q}}(T) = T + \frac{\sqrt{V(T)^2 + 4d(T)R_2(T)} - V(T)}{2R_2(T)}, \qquad \text{Eq. (33)}$$

$$f_{\mathrm{QP}}(T) = T + \min_{\mathbf{b}\in\mathcal{V}_2} \frac{\sqrt{V_{\mathbf{b}}(T)^2 + 4d_{\mathbf{b}}(T)R_{\mathbf{b}}(T)} - V_{\mathbf{b}}(T)}{2R_{\mathbf{b}}(T)}. \qquad \text{Eq. (37)} \tag{62}$$

The definition of the d, V, R functions can be found in the corresponding equations in the paper. To show all f functions are continuous, it suffices to have all d, V, R functions continuous. While d and R are continuous by their closed-form definition, the continuity of V requires slightly more attention. For example, by Equation (22), V seems to contain support mappings $\boldsymbol{s}_{\mathcal{B}}(\cdot)$ which is not continuous. However, V only depends on support functions in the form of $\boldsymbol{s}_{\mathcal{B}}(\boldsymbol{n})\cdot\boldsymbol{n} = \max_{\boldsymbol{b}\in\mathcal{B}}\boldsymbol{b}\cdot\boldsymbol{n}$, which is continuous in $\boldsymbol{n}$ when $\mathcal{B}$ is bounded and closed, and $\boldsymbol{n} = \boldsymbol{n}(T)$ is continuous in T. Therefore, V is also continuous in T. Similar results can be obtained for $V_{\boldsymbol{b}}$. Therefore, all f functions are continuous in T.

As shown in Section 4, we have

$$\begin{cases} T < f(T) \leq T^* & \text{if } 0 \leq T \leq T^*, \\ f(T) = T^* & \text{if } T = T^*, \end{cases} \tag{63}$$

where T^* denotes the TOI. Therefore, the sequence $\{T^{(k)}\}$ is strictly increasing and bounded above by T^*, which guarantees convergence to some limit $L \leq T^*$. Since f is also continuous on $[0, T^*]$, we have:

$$L = \lim_{k\to\infty} T^{(k+1)} = \lim_{k\to\infty} f(T^{(k)}) = f\Big(\lim_{k\to\infty} T^{(k)}\Big) = f(L). \tag{64}$$

Since $f(T) > T$ for any $T < T^*$, the only solution is $L = T^*$, i.e., $\lim_{k\to\infty} T^{(k)} = T^*$. This concludes that our CA-based iterative algorithm converges to the ground-truth TOI.

F GENERATION OF THE SYNTHETIC DATA

In this section, we provide details on the generation of the synthetic dataset in Section 6.1. The generation process of one data sample consists of two stages.

Stage 1: $\boldsymbol{V}, \boldsymbol{F}, \bar{\boldsymbol{A}}, \bar{\boldsymbol{v}}$. Two point sets $\boldsymbol{V}_1^{\mathrm{raw}}$ and $\boldsymbol{V}_2^{\mathrm{raw}}$, each with N_{raw} points, are generated from the standard normal distribution:

$$\boldsymbol{V}^{\mathrm{raw}} \sim \mathcal{N}_{N_{\mathrm{raw}}\times 3}(0,1). \tag{65}$$

We use the Eigen library to compute their convex hulls, meshes $(\boldsymbol{V}_1, \boldsymbol{F}_1)$ and $(\boldsymbol{V}_2, \boldsymbol{F}_2)$.

The initial transformation, $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{v}})$, is intentionally designed to be close to a rigid transformation, as is the case in ABD simulation in practice. We first sample a random rotation matrix $\boldsymbol{R}$ by performing singular value decomposition $\boldsymbol{R}^{\mathrm{raw}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ on a random matrix $\boldsymbol{R}^{\mathrm{raw}} \sim \mathcal{N}_{3\times 3}(0,1)$ and taking $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^\top$. We then add a small Gaussian noise onto $\boldsymbol{R}$ to obtain $\bar{\boldsymbol{A}} = \boldsymbol{R} + \boldsymbol{E}$, where $\boldsymbol{E} \sim \mathcal{N}_{3\times 3}(0, \sigma_{\bar{\boldsymbol{A}}} = 0.1)$. We sample $\bar{\boldsymbol{v}}$ uniformly inside the ball of radius $r_{\bar{\boldsymbol{v}}}$, centered at the origin. We choose $r_{\bar{\boldsymbol{v}}} = 2.0$ for $N_{\mathrm{raw}} = 10$ and $r_{\bar{\boldsymbol{v}}} = 3.0$ for $N_{\mathrm{raw}} = 256$.

We then compute the distance between the mesh $(\boldsymbol{V}_1, \boldsymbol{F}_1)$ transformed by $(\bar{\boldsymbol{A}}_1, \bar{\boldsymbol{v}}_1)$ and mesh $(\boldsymbol{V}_2, \boldsymbol{F}_2)$ transformed by $(\bar{\boldsymbol{A}}_2, \bar{\boldsymbol{v}}_2)$. If their distance is less than 10^{-9}, we reject this sample and restart from the beginning of Stage 1.

Stage 2: $\boldsymbol{A}, \boldsymbol{v}$. We first sample $\boldsymbol{A}^{\mathrm{raw}} \sim \mathcal{N}_{3\times 3}(0,1)$. We noted that the distribution of spectral norm of $\boldsymbol{A}^{\mathrm{raw}}$ is a bit concentrated around 2.2. To simulate scenarios with more diverse scales of $\boldsymbol{A}$, small or large, we rescaled $\boldsymbol{A}^{\mathrm{raw}}$ into $\boldsymbol{A} = s_{\boldsymbol{A}}\boldsymbol{A}^{\mathrm{raw}}/\|\boldsymbol{A}^{\mathrm{raw}}\|$, where $s_{\boldsymbol{A}} \sim \mathcal{U}(0, r_{\boldsymbol{A}} = 1.0)$ (the uniform distribution). Similarly, we sample $\boldsymbol{v} = s_{\boldsymbol{v}}\boldsymbol{v}^{\mathrm{raw}}/\|\boldsymbol{v}^{\mathrm{raw}}\|$, where $\boldsymbol{v}^{\mathrm{raw}} \sim \mathcal{N}_3(0,1)$ and $s_{\boldsymbol{v}} \sim \mathcal{U}(0, r_{\boldsymbol{v}} = 0.5)$.

We then compute the TOI using the generated shapes and motions. If the TOI is not between 0.01 and 1.0, we reject this sample and restart from the beginning of Stage 1.

G ADDITIONAL EXPERIMENTAL RESULTS

A Trajectory Example. Figure 11 illustrates the evolution of TOI estimates $\hat{T}$ and distances d during iterative CA updates for a representative trajectory. This example corresponds to the first synthetic data point (top-left) shown in Figure 4. Across all three C^{5}D variants, the lower-bound TOI estimates gradually converge toward the ground-truth TOI, while the distances between the two convex shapes decrease in a near-logarithmic manner. Among the methods, C^{5}D-Quad-PW exhibits the most efficient convergence in terms of iteration count for this example, consistent with the trends observed across the broader dataset summarized in Table 1.

Trade-off between CCD accuracy and ABD Newton iterations. Figure 12 illustrates how the maximum number of CCD iterations influences the convergence of the ABD solver. This experiment

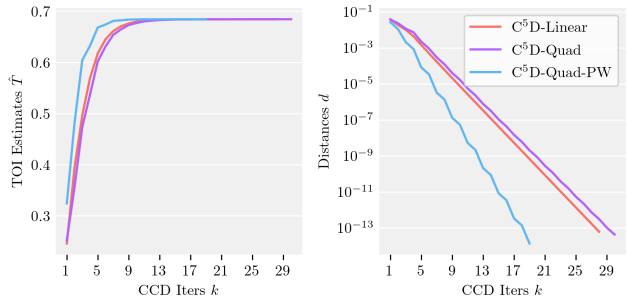

Fig. 11. **Trajectory Example:** Evolution of TOI estimates $\hat{T}$ and distances d during iterative CA updates. **Left:** Lower-bound TOI estimate progressively approaches the ground-truth TOI. **Right:** The distance between the two convex shapes decreases in a near-logarithmic manner.

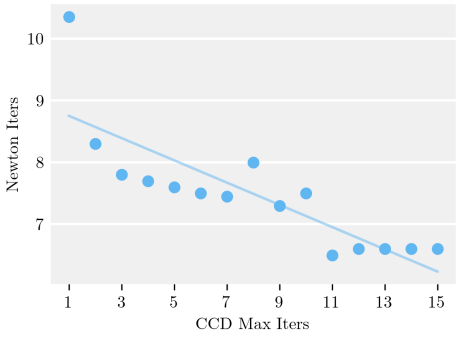

Fig. 12. **CCD–Newton Trade-Off**: Impact of the maximum number of CCD iterations on the convergence of the ABD solver. **X-axis**: maximum allowed CCD iterations per convex shape pair, with conservative advancement running until convergence or this limit. **Y-axis**: number of Newton iterations required for the ABD solver to converge.

uses the "Gears" scene (Figure 10) with a separation threshold $s = 0.25$. During each Newton iteration, conservative advancement is performed on each convex shape pair until either convergence is reached ($d < s\bar{d}$) or the iteration count hits `CCD_Max_Iters`. Varying `CCD_Max_Iters` impacts the number of Newton iterations required to meet the convergence criterion $\|\mathbf{p}\|_{\text{inf}} \leq 10^{-4}$, where $\mathbf{p}$ is the Newton update step.

The experiment demonstrates that increasing CCD accuracy reduces the number of Newton iterations required for convergence. When CCD is limited in accuracy–for example, with `CCD_Max_Iters` set to 1–the resulting step size in each Newton iteration becomes overly conservative, which in turn slows down the overall convergence of Newton's method.