Pacific Graphics 2015
N. J. Mitra, J. Stam, and K. Xu
(Guest Editors)

Volume 34 (2015), Number 7

TightCCD: Efficient and Robust Continuous Collision Detection using Tight Error Bounds

Zhendong Wang[1], Min Tang[1], Ruofeng Tong[1], and Dinesh Manocha[2,1]

http://gamma.cs.unc.edu/BSC/

[1]Zhejiang University, China
[2]University of North Carolina at Chapel Hill, USA

Abstract

We present a realtime and reliable continuous collision detection (CCD) algorithm between triangulated models that exploits the floating point hardware capability of current CPUs and GPUs. Our formulation is based on Bernstein Sign Classification that takes advantage of the geometry properties of Bernstein basis and Bézier curves to perform Boolean collision queries. We derive tight numerical error bounds on the computations and employ those bounds to design an accurate algorithm using finite-precision arithmetic. Compared with prior floating-point CCD algorithms, our approach eliminates all the false negatives and $90-95\%$ *of the false positives. We integrated our algorithm (TightCCD) with physically-based simulation system and observe speedups in collision queries of* $5-15X$ *compared with prior reliable CCD algorithms. Furthermore, we demonstrate its benefits in terms of improving the performance or robustness of cloth simulation systems.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics—Animation

1. Introduction

Collision detection is an important component of physically based simulation systems, including cloth, hair, and finite-element simulation. Many of these underlying simulation algorithms perform continuous collision detection (CCD), which tests for collisions between two discrete instances. Some of the more widely used algorithms in this regard are based on linearly interpolating motion between the vertices of each object. In that case, the CCD query between two triangles reduces to 15 elementary tests, each of which corresponds to finding the roots of a cubic polynomial [Pro97].

It is important to perform reliable CCD queries for many physically based simulations that can provide some accuracy guarantees on the underlying computations. It is well-known that even a single missed collision can affect the accuracy of the entire cloth simulation system [BFA02]. Most prior CCD algorithms are implemented using finite-precision arithmetic and this can result in two kinds of errors: *a false negative* that occurs when the CCD algorithm misses a collision; and *a false positive* that occurs when the CCD algorithm conservatively classifies a non-colliding instance as a collision.

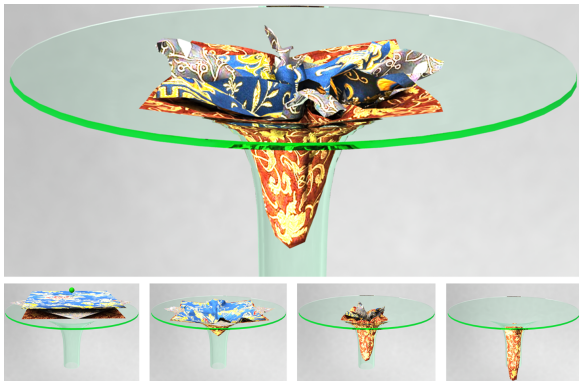


Figure 1: Funnel.*We use our CCD algorithm, TightCCD, for collision detection and response for cloth simulation. We highlight its performance on complex benchmarks with multiple layers. As compared to prior CCD algorithms, TightCCD improves the performance and reliability of the cloth simulation system.*

Many of these errors result when the underlying algorithm

use error tolerances along with floating-point arithmetic to perform elementary computations.

Many approaches have been proposed to overcome these accuracy or error tolerance problems that arise in the CCD computations. The most reliable algorithms are based on exact computations [BEB12] that can perform reliable queries with no false negatives or false positives. Recently, Tang et al. [TTWM14] presented another exact algorithm based on Bernstein sign classification that offers $10-20$X speedups over [BEB12]. Both these methods are based on an exact computation paradigm [Yap04] and use extended precision libraries to perform accurate CCD computations. In practice, these exact arithmetic operations can be expensive for real-time applications. Furthermore, it can be difficult to implement such exact arithmetic operations or libraries on GPUs or embedded processors. The second category of accurate solutions for CCD computations are based on performing floating-point error analysis and using appropriate error tolerances [Wan14]. This approach can be used on any processors that support IEEE floating-point arithmetic operations. The resulting CCD algorithm (SafeCCD) eliminates false negatives altogether but can still result in a high number of false positives, i.e. can be very conservative. These false positives can affect the performance and robustness of collision response computations.

Main Results: In this paper, we present a faster algorithm to perform robust CCD computations using floating point arithmetic. Our formulation is based on Bernstein sign classification (BSC) to perform CCD computations [TTWM14]. Instead of performing exact arithmetic operations for elementary tests, we perform a detailed error analysis on the underlying arithmetic operations and derive tight error bounds. These error bounds exploit the geometric properties of the Bézier curves and their control polygons. The overall algorithm (TightCCD) is simple and performs only addition, subtraction, multiplication, and comparison operations. Based on these error bounds, we can completely eliminate false negatives and significantly reduce the number of false positives. We have implemented our algorithm on CPUs and GPUs and highlight its performance for cloth simulation and finite-element (FEM) simulation. Compared with the original BSC algorithm [TTWM14], we observe $5-15$X speedup. Furthermore, our error bounds and the floating point algorithm yield a $90-95\%$ reduction in the number of false positives compared with a prior floating-point based accurate CCD algorithm [Wan14]. We also highlight the benefits of TightCCD in terms of improving the overall performance and reliability of cloth simulation systems.

Organization: The rest of the paper is organized as follows: We survey prior work on CCD computations in Section 2. We briefly review the BSC-based CCD algorithm in Section 3. We present our novel floating-point rounding error analysis of BSC in Section 4. We present our TightCCD algorithm in Section 5 and highlight its performance in Section 6.

2. Related Work

In this section, we offer a brief overview of prior work on CCD and high-level culling algorithms. There is extensive work on efficient CCD algorithms for different types of models. These include fast algorithms for rigid models [RKC02, KR03], articulated models [ZRLK07], and deformable models [VT94, GKJ*05, HF07, TMY*11] and most of them use the constant velocity assumption between the two discrete instances. All these algorithms perform 15 elementary tests for each triangle pair to compute the first time-of-contact using cubic root finding algorithms. Other classes of CCD algorithms for triangulated models are based on conservative local advancement [TKM09]. All these approaches are prone to floating-point errors and numerical tolerances. Recently, Wang [Wan14] provided a useful approach to improve their reliability based on forward error analysis for elementary tests. It includes derivation of tight error bounds for floating-point computation and these bounds are used to eliminate false negatives and reduce false positives. In many ways, we use similar ideas and derive tight error bounds on the BSC formulation for elementary tests [TTWM14]. We exploit more geometric properties of BSC to derive tighter error bounds than [Wan14] and highlight its benefits.

To reduce the number of CCD queries between triangle pairs, many high-level culling techniques have been proposed. The simplest culling algorithms compute geometric bounds using bounding volume hierarchies (BVHs) to reduce the number of false positives. Other culling methods are based on surface normal bounds [VT94, Pro97, MKE03], self-collision culling [SPO10, PKS10, ZJ12] and eliminating redundant tests between the vertices and edges of adjacent triangles [CTM08, TYM08, TCYM09, WB06]. Our reliable algorithm can be combined with any high-level culling algorithm as long as it can perform accurate computations.

3. BSC-based CCD

In this section, we briefly review the BSC-based CCD algorithm, and highlight some cases that can result in false negatives without exact geometric arithmetic.

3.1. CCD Formulation

Provot [Pro97] reduced CCD computation to coplanarity tests and inside tests. Tang et al. [TTWM14] combined the coplanarity and inside tests to find a common root of a system of algebraic equations and inequalities (i.e., a semi-algebraic set). Their approach employs properties of the Bernstein basis and Bézier curves and reduces elementary tests to performing a series of sign evaluations of algebraic expressions. Let's consider the Vertex/Face (VF) elementary test query. The same formulation is also used for the

Edge/Edge (EE) elementary test query. The VF query is reduced to checking whether the following semi-algebraic set has a real solution $t \in [0,1]$:

$$\begin{cases} Y(t) = 0 & (1) \\ G_1(t) \geq 0, G_2(t) \geq 0, G_3(t) \geq 0 & (2) \end{cases}$$

where $Y(t)$ is a cubic polynomial and $G_1(t), G_2(t), G_3(t)$ are three quartic polynomials. The equation (1) is a cubic equation that corresponds to the coplanarity test and the three inequalities (Equations (2)) are quartic equations that are used to perform the inside test.

3.2. Classification, Coplanarity Test and Inside Test

Classification: Before performing the coplanarity test, BSC-based CCD uses $Sign(Y'(0))$, $Sign(Y'(1))$, $Sign(Y''(0))$ and $Sign(Y''(1))$ to classify the cubic polynomial $Y(t)$ into three categories based on whether it has an inflection point or extreme point in domain [0,1]. $Y'(t)$ and $Y''(t)$ are its first order derivative and second order derivative, respectively. The operator $Sign()$ is used to compute the sign of a variable. $Sign(Y''(0)) \neq Sign(Y''(1))$ indicates an inflection point (shown in branch (a) of Figure 2); otherwise, there is no inflection point. Similarly, $Sign(Y'(0)) \neq Sign(Y'(1))$ indicates an extreme point (shown in branch (b) of Figure 2); otherwise, there is no extreme point.

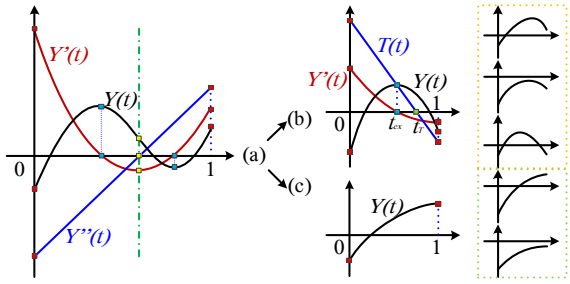
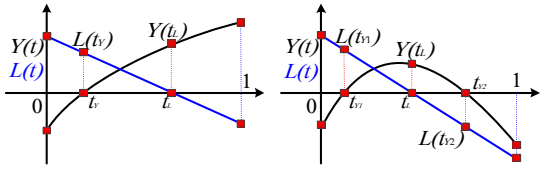


Figure 2: Classification and Coplanarity Test. *A cubic curve in branch (a) can be divided into two parts at its inflection point, and each part may correspond either to branch (b), where the curve has an extreme point, or branch (c). After classification, BSC-based CCD performs a coplanarity test for branches (b) and (c). Finally, there are five cases of roots.*

Coplanarity Test: If $Y(t)$ belongs to branch (b) in Figure 2 and $Sign(Y(0)) = Sign(Y(1))$ in the meantime, BSC-based CCD must compute the sign of $Y(t_{ex})$, in which t_{ex} is the root of $Y'(t)$, (i.e., the extreme point). If $Sign(Y(t_{ex})) \neq Sign(Y(0))$, there are two roots; otherwise, there is no root. The overall *Root-Finding Lemma* for this case is given in [TTWM14]. The coplanarity test for other situations is very simple: $Sign(Y(0)) \neq Sign(Y(1))$ means one root, and otherwise there is no root.

Inside Test: If $Y(t)$ has a root t_Y in domain [0,1] after the coplanarity test, the next step is to determine the sign of the quartic polynomial $G(t)$ at t_Y, i.e., $Sign(G(t_Y))$. Using the *Polynomial Decomposition Theorem* in [TTWM14], the inside test can be reduced to determine the signs of the two linear polynomials with the same constraint. Let $L(t)$ be one of the linear polynomials, $Sign(L(t_Y))$ can be computed by *Sign Determination Theorems I* and *II* in [TTWM14]. If all three inequalities corresponding to Equations (2) are true, along with the constraint of Equation (1), the inside test reports a collision.

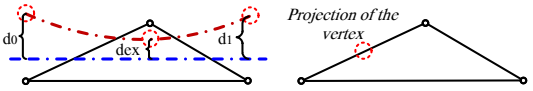


(a) One Root (b) Two Roots

Figure 3: Inside Tests. *(a) corresponds to Sign Determination Theorem I. (b) corresponds to Sign Determination Theorem II.*

3.3. Degenerate Cases

Without exact geometric arithmetic computation, $Sign()$ function is unable to compute an exact value because of floating-point rounding errors. This tends to happen more frequently in degenerate configurations. When continuous collision detection is used in physically based simulation, two typical degenerate cases arise. One corresponds to the fact when the distance between the features is very small. The other arises when a vertex is close to an edge of the triangle in terms of checking for coplanarity in a VF query.



(a) Small Distance (b) Close to Edge

Figure 4: Degenerate Cases in a VF test. *(a) The distances between the vertex and the face are very small. (b) During the coplanarity test, the vertex is close to one edge of the triangle.*

BSC-based CCD algorithm is also prone to these degenerate cases when it is implemented using floating-point operations (i.e. no exact computations).

- In the *Classification* computation, degenerate cases appear where $Y'(0) \approx 0, Y'(1) \approx 0, Y''(0) \approx 0$ and $Y''(1) \approx 0$, which may result in an incorrect classification and lead to errors in the coplanarity test.
- In the *Coplanarity test*, false negatives might arise due to small distances, such as $Y(0) \approx 0, Y(1) \approx 0$ and $Y(t_{ex}) \approx 0$.
- In the *Inside test*, a typical degenerate case involves a vertex that is close to an edge, which corresponds specifically to $t_L \approx t_Y$ in BSC-based CCD. In Figure 3, if $t_L \approx t_Y$, then

$Y(t_L) \approx 0$, which may result in imprecise $Sign(L(t_Y))$ and $Sign(G(t_Y))$.

4. Rounding Error Bounds

In this section, we present our novel floating-point rounding error analysis to derive a tight rounding error bound for floating-point operations. Eventually this bound is used to determine the sign of a floating-point number.

4.1. Sign Classification with Rounding Errors

One impact of rounding errors is that they may affect the sign of a floating-point number, as shown in Figure 5.

Given a floating-point number, let its exact value be r and its rounded floating-point value be $\hat{r}$. If σ_r is a tight bound on a corresponding rounding error, then $|\hat{r} - r| \leq \sigma_r$. The operator $Sign()$ returns the exact sign of a floating-point number because it uses an exact geometric computation paradigm. Using floating point hardware with rounding errors, we compute the sign of a floating-point number using the following rule:

$$Sign(r) = \begin{cases} 1, & \hat{r} \geq \sigma_r, \quad \text{i.e., } r \text{ is certainly positive;} \\ 0, & |\hat{r}| < \sigma_r, \quad r \text{ may be positive or negative;} \\ -1, & \hat{r} \leq -\sigma_r, \quad \text{i.e., } r \text{ is certainly positive.} \end{cases}$$

Thus, it is important to compute an accurate bound on the rounding error for sign classification.

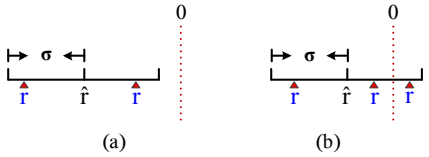


Figure 5: Rounding Errors. *The figure shows how rounding errors affect the sign of a floating-point number. In (a), $\hat{r} < -\sigma$, and $r < 0$. In (b), $-\sigma < \hat{r} < 0$, and $Sign(r)$ is undetermined. The case that $\hat{r} > 0$ can be derived similarly. We use these bounds in our reliable algorithm.*

4.2. Evaluating Error Bounds

In modern floating-point computers, the machine epsilon for a single precision floating-point number is $\varepsilon = 2^{-23}$. For a double precision floating-point number, it is $\varepsilon = 2^{-52}$.

Property of Rounding: Floating-point computation converts a real number r into a floating-point number $\hat{r}$ using a rounding process, and the relative rounding error satisfies the bound $\frac{|\hat{r}-r|}{|r|} < \varepsilon$, and $|r|\varepsilon$ is the corresponding rounding error bound. While using IEEE 754 standard, basic floating-point arithmetic operations are rounded so that such a bound holds. In other words, given an exact arithmetic operator $*$ and its floating-point counterpart $\circledast$, the condition $|a \circledast b - a * b| < |a * b|\varepsilon$ must be met [Wan14].

Rounding errors are accumulated when there are multiple floating-point operations, such as *add*, *subtract*, *multiply* and *divide*. In general, it is non-trivial to compute tight error bounds. In [Wan14], Wang proposed an error estimation theorem based on forward error analysis, which we refer to as *SafeBound* in the rest of the paper.

SafeBound: If f is a function made of *add*, *subtract*, and *multiply* operations, then $|f| \leq B_f$ and $|\hat{f} - f| \leq B_f[(1+\varepsilon)^{k_f} - 1]$, in which B_f and k_f are calculated using rules in Figure 6.

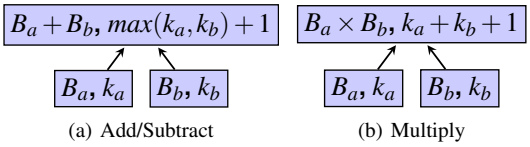


Figure 6: SafeBound. *A node is in the format of (B,k). If a floating-point number r has no rounding error, then $k_r = 0$.*

SafeBound is simple and efficient for deriving bounds on accumulated rounding errors, but the bounds increase rapidly (i.e. become conservative) when there are many operations, including subtraction involving two numbers with the same sign. In addition, it is not suitable to get a good bound for the *divide* operation. So we propose an alternate method to compute tighter rounding errors bound. To distinguish it from SafeBound, we refer to our novel approach as *TightBound*.

TightBound: Given two floating-point numbers, let their floating-point values be a and b, respectively. If both have no rounding errors, then $|(a \circledast b) - (a * b)| \leq |a * b|\varepsilon$. Because the exact value of $a * b$ is unknown, we can just evaluate the rounding error bound by its floating-point value $a \circledast b$. By extending the inequality properly,

$$-(|a \circledast b| + |a * b|\varepsilon) \leq a * b \leq |a \circledast b| + |a * b|\varepsilon, \quad (3)$$

it is easy to get $|a * b| \leq \frac{|a \circledast b|}{1-\varepsilon}$. So the *tightest* rounding error bound for this single operation is $\sigma_{(a*b)} = |a \circledast b| \frac{\varepsilon}{1-\varepsilon}$ (shown in Figure 7(a)).

If the two numbers have rounding errors, let σ_a and σ_b be the corresponding bounds on rounding errors, respectively. We consider various *add*, *subtract*, *multiply* and *divide* operations in following theorem.

Theorem 1: *Let σ_{a*b} be a tight bound on rounding error of $a * b$, where $*$ could be* add, subtract, multiply *and* divide *operation. σ_{a*b} is computed by rules in Figure 7.*

The proof can be found in Appendix.

Let σ^T be the bound derived from TightBound and σ^S be the bound derived from SafeBound, respectively. We can compare TightBound with SafeBound based on the following theorem.

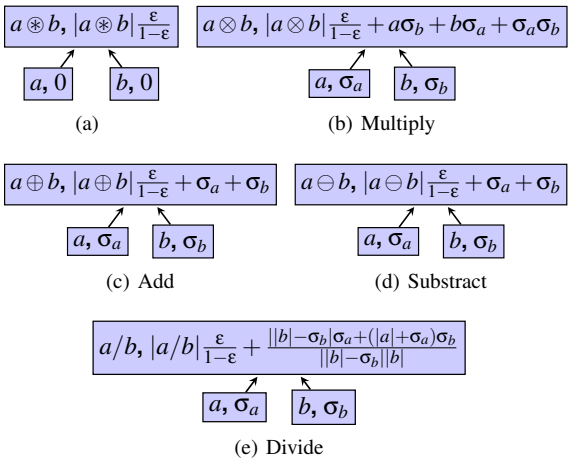


Figure 7: TightBound. *A floating-point number is in the format of (L,R), where L represents its floating-point value and R represents its rounding error bound. $R = 0$ indicates that there is no rounding error. $\circledast$ represents floating-point arithmetic operations (add, subtract, multiply). We use $\square/\square$ to represent floating-point division, and $\frac{\square}{\square}$ to represent exact division.*

Theorem 2: *Given a floating-point expression exp, we get:* $\sigma^T_{exp} \leq \sigma^S_{exp}$.

Let B_a and B_b be bounds for a and b, both having no rounding errors, in SafeBound, respectively. To remain safe, it must satisfy $B_a + B_b \geq max(\frac{|a \oplus b|}{1-\varepsilon}, \frac{|a \ominus b|}{1-\varepsilon})$ and $B_a \times B_b \geq \frac{|a \otimes b|}{1-\varepsilon}$. In general, the bound in SafeBound for an arbitrary floating-point number r must be no less than $\frac{|r|}{1-\varepsilon}$, i.e., $B_r \geq \frac{|r|}{1-\varepsilon}$. In contrast, Wang [Wan14] uses a large uniform bound for all floating-point variables involved in computation.

Clearly, $\sigma^T_{a \circledast b}$ is tighter than the $\sigma^S_{a \circledast b}$ if two floating-point numbers a and b have no rounding error. Based on this point, we can suppose that $\sigma^T_a \leq \sigma^S_a$ and $\sigma^T_b \leq \sigma^S_b$ if a and b are with rounding errors. And it is easy to get the inference that $\sigma^T_{a \circledast b} \leq \sigma^S_{a \circledast b}$. The detail derivation process can be found in Appendix. According to mathematical induction, we can conclude that $\sigma^T_{exp} \leq \sigma^S_{exp}$ for a floating-point expression.

5. TightCCD

In this section, we present our TightCCD algorithm, which is the combination of TightBound and BSC-based CCD algorithm. We use the new $Sign()$ operator with tight rounding error bounds derived in Section 4 to capture degenerate cases existing in a BSC-based CCD algorithm. By performing certain conservative operations in degenerated cases, we ensure that our TightCCD algorithm remains reliable using only floating-point arithmetic. In addition, we can control the number of false positives at a very low level based on certain tight bounds on rounding errors.

5.1. Bounds for Critical Variables

In a coplanarity test, the cubic polynomial in Equation (1) can be formulated in terms of the following Bernstein formulation: $Y(t) = k_0 * B^3_0(t) + k_1 * B^3_1(t) + k_2 * B^3_2(t) + k_3 * B^3_3(t)$, where $B^3_i(t)$ is the cubic Bernstein basis. The derivation of k_0, k_1, k_2 and k_3 is given in [TTWM14]. Let $\sigma_{k_0}, \sigma_{k_1}, \sigma_{k_2}$ and σ_{k_3} be the bounds on the rounding errors in these quantities, respectively. Given these formulas to compute the derivatives of $Y()$, we compute the rounding error bounds for $Y(0)$, $Y(1)$, $Y'(0)$, $Y'(1)$, $Y''(0)$ and $Y''(1)$. According to our rounding error formulation (TightBound), we obtain the bounds shown in the following table. The rounding error bounds for other intermediate variables can be derived similarly and also based on the TightBound.

Variables	Values	Bounds
$Y(0)$	k_0	σ_{k_0}
$Y(1)$	k_0	σ_{k_3}
$Y'(0)$	$3(k_1 - k_0)$	$6\lvert k_1 - k_0\rvert \frac{\varepsilon}{1-\varepsilon} + 3\sigma_{k_1} + 3\sigma_{k_0}$
$Y'(1)$	$3(k_3 - k_2)$	$6\lvert k_3 - k_2\rvert \frac{\varepsilon}{1-\varepsilon} + 3\sigma_{k_2} + 3\sigma_{k_3}$
$Y''(0)$	$6(k_2 - 2k_1 + k_0)$	$(12\lvert k_2 - 2k_1 + k_0\rvert + 6\lvert k_2 - 2k_1\rvert + 12\lvert k_1\rvert) \frac{\varepsilon}{1-\varepsilon} + 6(\sigma_{k_0} + \sigma_{k_2} + 2\sigma_{k_1})$
$Y''(1)$	$6(k_3 - 2k_2 + k_1)$	$(12\lvert k_3 - 2k_2 + k_1\rvert + 6\lvert k_3 - 2k_2\rvert + 12\lvert k_2\rvert) \frac{\varepsilon}{1-\varepsilon} + 6(\sigma_{k_3} + \sigma_{k_1} + 2\sigma_{k_2})$

Although bounds derived from SafeBound [Wan14] are not shown here, we use two statistical charts (Figure (8)) to demonstrate that the bounds of these variables derived from TightBound are tighter than their bounds derived from SafeBound.

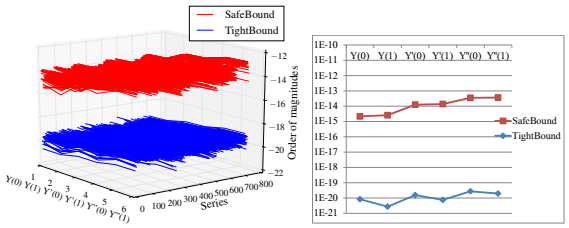


Figure 8: Comparison of TightBound and SafeBound: *(a) illustrates the bounds derived using different approaches for the six critical variables in a BSC-based CCD algorithm; (b) shows the detailed bounds for a single collision test. This figure shows that the order of magnitudes of bounds derived from TightBound are roughly* $10e-20$*, while that of bounds derived from SafeBound are roughly* $10e-14$.

5.2. Conservative Operations

Classification: In classification, using our new $Sign()$ with rounding error bounds, $Sign(Y''(0)) = 0$, $Sign(Y''(1)) = 0$, $Sign(Y'(0)) = 0$ or $Sign(Y'(1)) = 0$ may happen. If these

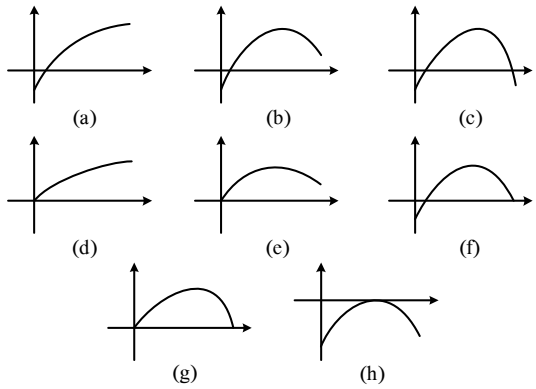


Figure 9: Classification Based on the Roots of Y(t). *In (a),(b),(d),(e), $Y(t)$ has only one root in domain $[0,1]$. In (c),(f),(g),(h), $Y(t)$ has two roots in domain $[0,1]$. In (d) and (e), $Sign(Y(0)) = 0$. In (f), $Sign(Y(1)) = 0$. In (g), $Sign(Y(0)) = 0$ and $Sign(Y(1)) = 0$. In (h), the sign of the extreme point is undetermined. All small distance cases are to be captured.*

zero signs are not handled properly, the classification may miss an inflection point or an extreme point, which would yield false negatives to CCD. Thus, we perform two types of conservative operations as follows:

1. when $Sign(Y''(0)) = 0$ or $Sign(Y''(1)) = 0$, there is an inflection point in the domain $[0,1]$.
2. when $Sign(Y'(0)) = 0$ or $Sign(Y'(1)) = 0$, there is an extreme point in the domain $[0,1]$.

Coplanarity Test: In a coplanarity test, the zero signs must be handled properly. If there is no extreme point in domain [0,1], we conservatively report a root when $Sign(Y(0)) = 0$ or $Sign(Y(1)) = 0$. A challenging case occurs when there is an extreme point in domain [0,1] meanwhile $Sign(Y(0)) = 0$ or $Sign(Y(1)) = 0$. To use the *Root-Finding Lemma* to handle this case, we must preprocess the signs of certain critical variables.

1. Ensure $Sign(Y(0)) = Sign(Y(1))$.
   - If $Sign(Y(0)) = Sign(Y(1)) = 0$, then simply report two roots.
   - If $Sign(Y(0)) = 0$ and $Sign(Y(0)) \neq 0$, then set $Sign(Y(0)) = Sign(Y(1))$.
2. Ensure $Sign(Y'(0)) \neq Sign(Y'(1))$.
   - If $Sign(Y'(0)) = 0$ and $Sign(Y'(1)) \neq 0$, then set $Sign(Y'(0)) = -Sign(Y'(1))$.
   - The troublesome case is when $Sign(Y'(0)) = Sign(Y'(1)) = 0$. Using $Y'(t)$ to decompose $Y(t)$, we obtain $Y(t) = Y'(t) \cdot S(t) + T(t)$ in which $S(t)$ and $T(t)$ are two linear polynomials. In this case, $Y(t) \approx T(t)$ in domain [0,1]. If $Sign(T(0)) = Sign(T(1)) = 0$, then simply report two roots. Otherwise, the sign of the extreme point is equal to the sign of $T(0)$, i.e., $Sign(Y(t_{ex})) = Sign(T(0))$.

After completing the computation described above, the instance that we need to address corresponds to branch (b) in Figure 2. In this case, the signs of all the pivotal variables are non-zero. The overall procedure of root finding for this case is given in Algorithm. 1. Figure 9 shows that there are eight different classifications of roots of $Y(t)$ after the coplanarity test. We must mainly address (a), (b) and (c) in our modified inside test algorithm. By simply setting $Sign(Y(0)) = -Sign(Y(1))$, (d) can be transformed to (a) and (e) can be transformed to (b). Similarly, (f), (g) and (h) can be transformed to (c).

Algorithm 1 Coplanarity Test with Conservative Operations

```
1: Input: Y(t)      Output: Number of roots.
2: Ensure Sign(Y(0)) = Sign(Y(1)).
3: Ensure Sign(Y'(0)) ≠ Sign(Y'(1)).
4: Root-Finding Lemma. [TTWM14]
5: if Y(t) has no root then
6:    if Sign(Y(0)) = 0 or Sign(Y(1)) = 0 then
7:       return 1; //Conservatively
8:    end if
9: end if
```

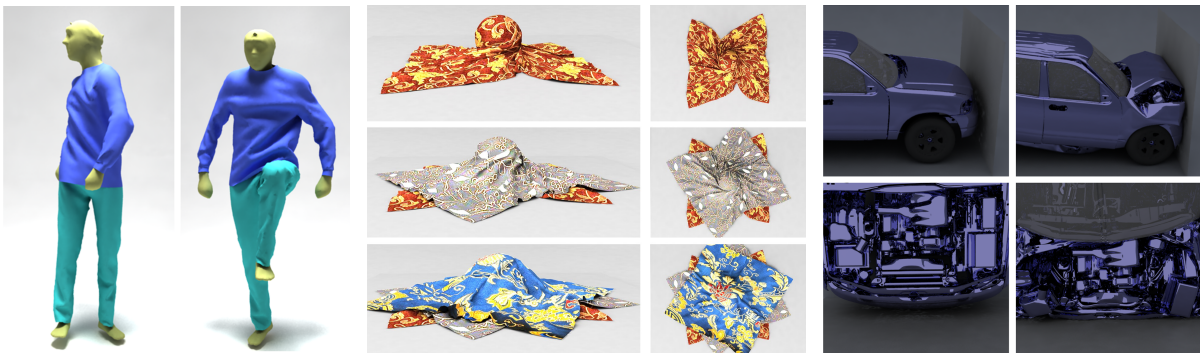
Inside Test: In the algorithm for the inside test, we combine conservative operations with *Sign Determination Theorems I* and *II*, as shown in Algorithm 2. In fact, we need only to take account of the case in which $Sign(Y(t_L))) = 0$, where t_L is the root of the linear polynomial $L(t)$ (shown in Figure 3). $Sign(Y(t_L))) = 0$ implies t_L is very close to one root of $Y(t)$ and also corresponds to the degenerate case of *Close to Edge*. In this case, we use some conservative operations to overcome these errors.

1. Case one: $Y(t)$ has only one root. $Sign(L(t_Y)) \leftarrow 0$.
2. Case two: $Y(t)$ has two roots.
   - If $Sign(Y'(t_L)) = 0$, then $Sign(L(t_{Y_0})) \leftarrow 0$ and $Sign(L(t_{Y_1})) \leftarrow 0$.
   - If $Sign(Y'(t_L)) = Sign(Y'(0))$, then $Sign(L(t_{Y_0})) \leftarrow 0$ and $Sign(L(t_{Y_1})) \leftarrow Sign(L(1))$.
   - If $Sign(Y'(t_L)) = Sign(Y'(1))$, then $Sign(L(t_{Y_0})) \leftarrow Sign(L(0))$ and $Sign(L(t_{Y_1})) \leftarrow 0$.

Finally, we obtain the signs of three quartic polynomials under the condition of a cubic equation. If one of their signs is definitely negative, we can conclude that there is no collision. Otherwise, we report a collision, which may also turn out to be a false positive.

5.3. False Positives

Since we have introduced rounding error bounds and conservative operations into BSC-based CCD algorithm, our approach may report false positives, especially for degenerate



(a) Virtual Clothes (b) Twisting (c) Car

Figure 10: Benchmarks: *We use these cloth and FEM simulation benchmarks to evaluate the performance of our CCD algorithm. TightCCD can perform reliable collision queries in these benchmarks.*

Algorithm 2 Inside Test with Conservative Operations

Input: $G(t), Y(t_Y) = 0$ *//$G(t)$ is quartic, $Y(t)$ is cubic.*
Output: $Sign(G(t_Y))$
Using *Polynomial Decomposition Theorem* to get two linear polynomials $L_1(t)$ and $L_2(t)$.
for $i = 1,2$ **do**
  if $Sign(Y(t_{L_i})) = 0$ **then**
    Conservative operations;
  else
    Sign Determination Theorems I or *II*. [TTWM14]
  end if
end for
$Sign(G(t_Y)) = Sign(L_1(t_Y))Sign(L_2(t_Y))$.

cases. In coplanarity tests with conservative operations, *Small Distance*, shown as Figure 9(d), (e), (f), (g), (h), may also result in false positives. In the Inside Test with conservative operations, *Close to Edge*, which corresponds to $Sign(Y(t_L)) = 0$, may also contribute to false positives.

6. Results

We use our tight error bounds to perform conservative operations in three stages of a BSC-based CCD algorithm: classification, the coplanarity test and the inside test. Implementing these modified algorithms is simple and involves only a few operations and comparisons. The resulting algorithm (TightCCD) is implemented using IEEE double precision arithmetic. Compared with the original BSC-based CCD algorithm [TTWM14], TightCCD results in $5-15$X speedup on a single core. However, the original algorithm can provide exact results on the CPU using an extended precision library, whereas our formulation is slightly conservative and can return false positives. Compared with the SafeCCD algorithm that also uses error bounds [Wan14], TightCCD is slightly slower but can reduce the number of false positives by $90-95\%$ because we use tighter error bounds.

6.1. Performance

We evaluated the performance of TightCCD with some cloth simulations and FEM simulation benchmarks, as shown in Figure 10, using a standard PC (Intel i7-4770 CPU @3.4GHz, 4 GB RAM, 64-bits Windows 7 OS, NVIDIA GeForce GTX 780 GPU). This includes a CPU-based C++ implementation of BSC-exact [TTWM14] that uses TightBounds and IEEE double precision arithmetic. Figure 11 highlights the performance of TightCCD algorithms on different benchmarks, on CPUs and GPUs. We also compare the performance of TightCCD with the following algorithms and implementation:

- **BSC** This is the implementation of the exact algorithm of [TTWM14], published by the authors (`http://gamma.cs.unc.edu/BSC/`). It uses a conservative culling test to accelerate the computation. There are two version of BSC, **BSC-float** and **BSC-exact**. *BSC-float* is based on floating-point arithmetic so it may report both false positives and false negatives. In contrast, *BSC-exact* can eliminate both false positives and false negatives because it uses interval arithmetic-based filters and exact expansions for exact arithmetic operations.
- **SafeCCD:** This is the implementation of the floating-point-based cubic root solver CCD algorithm [Wan14] with error analysis. (`http://web.cse.ohio-state.edu/~whmin/Wang-2014-DCC/SAFE\_CCD.zip`). It uses a forward rounding error to compute bounds and (SafeBound in Section 4.2) to ensures that no false negatives are reported. However, it can result in false positives.

Furthermore, we make some modifications to *BSC-float* based on our error bound computation (TightBound). We use these conservative operations to eliminate false negatives

Benchmarks	# of Collision Queries	BSC-exact		TightCCD			BSC with SafeBound			SafeCCD with TightBound			SafeCCD (with SafeBound)			BSC-float		
Virtual Clothes	2.86 M	7.47	0	1.24	0.35	14751	1.42	0.37	57721	3.43	0.63	2341	0.8	0.16	129259	0.14	2512	1194
Twisting	5.75 M	4.1	0	0.88	0.24	90046	1.02	0.27	1703728	4.33	0.65	2103	0.67	0.16	1706630	0.13	2070	14969
Funnel	6.6 M	11.2	0	0.73	0.17	7490	0.86	0.2	728351	3.45	1.34	40	0.61	0.22	126253	0.13	2411	1178
Car	8.19 M	8.2	0	0.62	0.19	1005	0.81	0.21	230475	4.2	1.92	114	0.62	0.25	37677	0.13	1375	5546

Unit of Time: *us* — Average time of CCD Query on CPU — Average time of CCD Query on GPU — Number of False Positives — Number of False Negatives

Figure 11: Performance and Comparison: *We highlight the performance of six different CCD algorithms on several different benchmarks on a single CPU core and a Nvidia GeForce GTX 780 GPU. BSC-exact only runs on the CPU and it can eliminate both false negatives and false positives. In contrast, BSC-float runs on GPUs and may result in false negatives and false positives, as it uses no error bounds. The other four algorithm are based on error bounds, so that they can eliminate false negative but report some false positives. We also replaced the error bound computation in the SafeCCD algorithm with TightBounds and observe significant reduction in the number of false positives. Compared with BSC-exact, TightCCD is* $5-15X$ *faster. As compared with SafeCCD, TightCCD is less conservative, i.e. reports fewer false positives.*

and report as few false positives as possible based on tight rounding error bounds. As a result, we obtain our **TightCCD** algorithm, a new version of the BSC. Furthermore, we also use the rounding error estimation theorem proposed by [Wan14] (SafeBound) to produce another version of the BSC called **BSC with SafeBound**, which also reports no false negatives but more false positives than *TightCCD*. We also try our TightBound in *SafeCCD*, which results in **SafeCCD with TightBound**, another version of *SafeCCD*.

Figure 11, we can observe that BSC-exact algorithm can eliminate false positives by using exact geometric arithmetic, while the other four algorithms (including our *TightCCD* algorithm) can yield false positives. We observe appreciable speedups using our BSC with rounding error bounds based on floating-point arithmetic (i.e. TightCCD) vs the BSC-exact CCD algorithm with exact geometric arithmetic. In the benchmarks, *Virtual Cloth* and *Twisting* our *TightCCD* is a little slower than *SafeCCD*, but *TightCCD* results in significantly fewer false positives. Although SafeBound is also suitable for BSC-based CCD, *BSC with SafeBound* reports more false positives and is little slower than *TightCCD*. In addition, TightBound can also be combined with the *SafeCCD* algorithm. On one hand, we observe *SafeCCD with TightBound* reports the minimum number of false positives among the four conservative algorithms; on the other hand, it is the slowest amongst the conservative algorithms implemented using floating-point arithmetic. By using our TightBound in the benchmark *Virtual Cloth*, *TightCCD* reduces the false positives by 90% and *SafeCCD with TightBound* reduces the false positives by almost 99%, compared with the original *SafeCCD* algorithm [Wan14]. In the benchmarks *Twisting*, *Funnel* and *Car*, TightCCD reduces the number of false positives by 95% and more.

Figure 11 also shows the performance of five floating-point arithmetic based CCD algorithms on a GPU. We observe that *BSC-float* reports some false negatives without rounding error bounds, while the other four algorithms with rounding error bounds can eliminate false negatives. The false positives reported by these four algorithms on a GPU are the same as those reported on the CPU. Due to the parallelism of a GPU, CCD algorithm implementations on the GPU are much faster than CPU-based implementations.

6.2. Cloth Simulation: Overall Benefits

We integrated *TightCCD, SafeCCD* and *BSC-exact* into the same cloth simulation system and use them for collision detection. As the number of false positives increase, this results in more time spent in the collision response computation. We observe different performance of resulting cloth simulation by based on three CCD algorithms. We use the benchmark *Funnel* with multiple layers of cloth to highlight the differences in the performance. In Figure 12, we observe two main benefits of *TightCCD* over *SafeCCD* and *BSC-exact*. .

- Faster collision detection: TightCCD is much faster then BSC-exact. This results in less time spent in collision checking with TightCCD. On the other hand, the fraction of time spent in collision handling with BSC-exact is much higher..
- Faster and reliable collision response: As collisions are detected, the cloth simulation algorithm responds to each collision, until those triangles are in a non-penetrating state. If the underlying collision detection algorithm is very conservative, i.e. reports many false positives, the collision response algorithm needs to perform many additional iterations to compute non-penetrating configurations.

Funnel Benchmarks	Frames	TightCCD				BSC-exact				SafeCCD				
One Layer	0-200	1	0.53	0.15	0.69	1	0.53	0.15	0.7	1	0.52	0.14	0.68	Fail to Converge at Frame 1189
	200-700	1.06	0.69	0.34	1.05	1.06	0.69	0.44	1.15	1.06	0.68	0.26	1	
	700-1330	2.99	0.87	0.53	1.42	3.2	0.88	1.25	2.15	3.22	0.85	0.61	1.49	
Two Layers	0-200	1	1.04	0.28	1.34	1	1.02	0.28	1.33	1	1.05	0.28	1.35	Fail to Converge at Frame 229
	200-700	1.41	1.58	0.88	2.47	1.46	1.56	1.17	2.76	1.57	1.22	0.41	1.65	
	700-1330	3.53	2	1.5	3.52	3.63	1.97	3.94	5.94	/	/	/	/	
Three Layers	0-200	1	1.52	0.39	1.93	1	1.53	0.39	1.94	1	1.53	0.39	1.93	Fail to Converge at Frame 229
	200-700	2.19	2.55	1.47	4.03	2.2	2.57	1.5	4.1	1.61	1.7	0.52	2.24	
	700-1330	4.75	3.41	3.09	6.52	4.73	3.33	13.3	16.7	/	/	/	/	

Unit of Time: *s* — Average number of iterations in collision response computation; Average time spent in the integration step; Average time spent in collision detection and response; Average frame time

Figure 12: Benefits for Cloth Simulation: *We highlight the performance and reliability of cloth simulation system based on three CCD algorithms: TightCCD, BSC-Exact, and SafeCCD. All the frame timings are reported in seconds. TightCCD and BSC-exact are able to perform reliable collision detection and response computations. However, we observe up to 2.5X improvement in the average frame time with TightCCD. In contrast, we observe some convergence problems (for few frames) with SafeCCD. Due to a high number of false positives in SafeCCD, we observe slower convergece (or failed convergence) in terms of collision response computations. Overall, TightCCD offers the best overall performance and reliability.*

Figure 12 highlights the extra complexity of collision response due to higher number of false positives, i.e. additional iterations and no convergence in some frames. We observe this behavior with SafeCCD in our benchmarks.

6.3. Analysis

The notion of computing rounding errors is quite common in scientific and geometric algorithms and has been brought to the forefront by Wang [Wan14] with the SafeCCD algorithm. We present a modified algorithm based on BSC that computes tighter error bounds. In particular, we compute bounds on the values of each variable involved in that algorithm and use that for a modified coplanarity test and inside test. Our main advantage is that we can compute tighter error bounds. However, a slightly higher computational overhead is involved computing these bounds on the rounding errors. If there is no collision, the higher level culling algorithm is able to discard most of those instances. The worst-case arises when the cubic polynomial in a coplanarity test has an inflection point. SafeCCD derives a formulation for the rounding error bound for every intermediate variable, which leads to a faster algorithm. Its drawback is that the rounding error bounds are not very tight, and therefore there are more false positives. We have highlighted the benefits in terms of improved performance and reliability of the resulting cloth simulation system.

7. Limitations, Conclusions and Future Work

We present a reliable algorithm for CCD computation that is based on computing tight bounds on forward rounding errors. We apply those bounds to various steps of the BSC-based CCD algorithm and perform all the computations using IEEE floating point arithmetic. The modified algorithm performs conservative computations, which can result in some false positives. We highlight its benefits compared with prior reliable CCD algorithms and also demonstrate its benefits on the cloth simulation system.

Our approach has a few *limitations*. Due to conservative computations, our modified BSC-based CCD can result in false positives and is not exact. Moreover, its computational overhead is slightly greater than the approach presented by Wang [Wan14].

There are many avenues for future work. It may be possible to make our approach less conservative. Furthermore, it can be parallelized using the SIMD capabilities of GPUs and used for other physically based simulation applications. Finally, we would use TightCCD for other dynamic simulation systems.

Acknowledgement

This research is supported in part by NSFC (61170140), the National High-Tech Research and Development Program of China (No.2013AA013903), the National Key Technology R&D Program of China (2012BAD35B01), the Doctoral Fund of Ministry of Education of China (20130101110133). Dinesh Manocha is supported in part by ARO Contract W911NF-10-1-0506, Intel and the Office Of The Director, National Institutes Of Health under Award Number R44OD018334, and the National Thousand Talents Program of China. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. Ruofeng Tong is partly supported by NSFC (61170141).

References

[BEB12] BROCHU T., EDWARDS E., BRIDSON R.: Efficient geometrically exact continuous collision detection. *ACM Trans. Graph. 31*, 4 (July 2012), 96:1–96:7. 2

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (July 2002), 594–603. 1

[CTM08] CURTIS S., TAMSTORF R., MANOCHA D.: Fast collision detection for deformable models using representative-triangles. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (2008), pp. 61–69. 2

[GKJ*05] GOVINDARAJU N. K., KNOTT D., JAIN N., KABUL I., TAMSTORF R., GAYLE R., LIN M. C., MANOCHA D.: Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24*, 3 (July 2005), 991–999. 2

[HF07] HUTTER M., FUHRMANN A.: Optimized continuous collision detection for deformable triangle meshes. In *Proceedings of WSCG* (2007), pp. 25–32. 2

[KR03] KIM B., ROSSIGNAC J.: Collision prediction for polyhedra under screw motions. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications* (2003), SM '03, pp. 4–10. 2

[MKE03] MEZGER J., KIMMERLE S., ETZMUSS O.: Hierarchical techniques in cloth detection for cloth animation. *Journal of WSCG 11* (2003), 322–329. 2

[PKS10] PABST S., KOCH A., STRASSER W.: Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. *Computer Graphics Forum 29*, 5 (July 2010), 1605–1612. 2

[Pro97] PROVOT X.: Collision and self-collision handling in cloth model dedicated to design garments. *Graphics Interface* (1997), 177–189. 1, 2

[RKC02] REDON S., KHEDDAR A., COQUILLART S.: Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 2*, 3, SI (2002), 279–287. 2

[SPO10] SCHVARTZMAN S. C., PÉREZ A. G., OTADUY M. A.: Star-contours for efficient hierarchical self-collision detection. *ACM Trans. Graph. 29*, 4 (July 2010), 80:1–80:8. 2

[TCYM09] TANG M., CURTIS S., YOON S.-E., MANOCHA D.: ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (Aug. 2009), 544–557. 2

[TKM09] TANG M., KIM Y. J., MANOCHA D.: C^2A: Controlled conservative advancement for continuous collision detection of polygonal models. In *Proceedings of IEEE International Conference on Robotics and Automation* (May 2009), ICRA'2009, pp. 356–361. 2

[TMY*11] TANG M., MANOCHA D., YOON S.-E., DU P., HEO J.-P., TONG R.-F.: VolCCD: Fast continuous collision culling between deforming volume meshes. *ACM Trans. Graph. 30*, 5 (Oct. 2011), 111:1–111:15. 2

[TTWM14] TANG M., TONG R., WANG Z., MANOCHA D.: Fast and exact continuous collision detection with bernstein sign classification. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 186:1–186:8. 2, 3, 5, 6, 7

[TYM08] TANG M., YOON S.-E., MANOCHA D.: Adjacency-based culling for continuous collision detection. *The Visual Computer 24*, 7 (2008), 545–553. 2

[VT94] VOLINO P., THALMANN N. M.: Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum 13*, 3, CI (Sept. 1994), 155–166. 2

[Wan14] WANG H.: Defending continuous collision detection against errors. *ACM Trans. Graph. 33*, 4 (July 2014), 122:1–122:10. 2, 4, 5, 7, 8, 9

[WB06] WONG W. S.-K., BACIU G.: A randomized marking scheme for continuous collision detection in simulation of deformable surfaces. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications* (2006), VRCIA '06, pp. 181–188. 2

[Yap04] YAP C.: *Robust geometric computation*. 2004. In Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, Eds., 2nd ed. Chapmen & Hall/CRC, Boca Raton, FL, ch. 41. 2

[ZJ12] ZHENG C., JAMES D. L.: Energy-based self-collision culling for arbitrary mesh deformations. *ACM Trans. Graph. 31*, 4 (July 2012), 98:1–98:12. 2

[ZRLK07] ZHANG X., REDON S., LEE M., KIM Y. J.: Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Trans. Graph. 26*, 3 (July 2007). 2

Appendix

In this section, we present some derivations and proofs.

Derivation of TightBound

If the two numbers a and b have rounding errors, let $\hat{a}$ and $\hat{b}$ be their rounded floating-point values and σ_a and σ_b be the corresponding bounds on rounding errors, respectively. We consider various *add*, *subtract*, *multiply* and *divide* operations in following theorem.

Theorem 1: *Let σ_{a*b} be a tight bound for $a*b$, where $*$ could be* add, subtract, multiply *and* divide *operation.*

Add: $\sigma_{a+b} = |\hat{a}\oplus\hat{b}|\frac{\varepsilon}{1-\varepsilon}+\sigma_a+\sigma_b$ (shown in Figure 7(c)). $|(\hat{a}\oplus\hat{b})-(a+b)| \le |(\hat{a}\oplus\hat{b})-(\hat{a}+\hat{b})|+|\hat{a}-a|+|\hat{b}-b| \le |\hat{a}\oplus\hat{b}|\frac{\varepsilon}{1-\varepsilon}+\sigma_a+\sigma_b$.

Subtract: *subtract* are similar to *add* in terms of error bounds. We could similarly obtain $\sigma_{(a-b)} = |\hat{a}\ominus\hat{b}|\frac{\varepsilon}{1-\varepsilon}+\sigma_a+\sigma_b$ (shown in Figure 7(d)).

Multiply: $\sigma_{a\times b} = |\hat{a}\otimes\hat{b}|\frac{\varepsilon}{1-\varepsilon}+|\hat{a}|\sigma_b+\sigma_a|\hat{b}|+\sigma_a\sigma_b$ shown in Figure 7(b)). $|(\hat{a}\otimes\hat{b})-(a\times b)| \le |(\hat{a}\otimes\hat{b})-(\hat{a}\times\hat{b})|+|\hat{a}\times(\hat{b}-b)|+|\hat{b}\times(\hat{a}-a)|+|\hat{b}-b|\times|\hat{a}-a| \le |\hat{a}\otimes\hat{b}|\frac{\varepsilon}{1-\varepsilon}+|\hat{a}|\sigma_b+\sigma_a|\hat{b}|+\sigma_a\sigma_b$.

Divide: $\sigma_{\frac{a}{b}} = |\hat{a}/\hat{b}|\frac{\varepsilon}{1-\varepsilon}+\frac{||\hat{b}|-\sigma_b|\sigma_a+(|\hat{a}|+\sigma_a)\sigma_b}{||\hat{b}|-\sigma_b||\hat{b}|}$ (shown in Figure 7(e)). $|\hat{a}/\hat{b}-\frac{a}{b}| \le |\hat{a}/\hat{b}-\frac{\hat{a}}{\hat{b}}|+|\frac{\hat{a}}{\hat{b}}-\frac{a}{\hat{b}}|+\frac{|a(b-\hat{b})|}{|b\hat{b}|} \le |\hat{a}/\hat{b}|\frac{\varepsilon}{1-\varepsilon}+\frac{\sigma_a}{|\hat{b}|}+\frac{(|\hat{a}|+\sigma_a)\sigma_b}{||\hat{b}|-\sigma_b||\hat{b}|}$. (It is only applicable for the case when $|\hat{b}|>\sigma_b$.)

Comparison between TightBound and SafeBound

Because SafeBound is not suitable for *divide* operations, we take consideration only of *add*, *subtract* and *divide* operations in the following comparison. Let σ^T be the bound

derived by TightBound and σ^S be the bound derived by SafeBound, respectively.

Theorem 2: *Given a floating-point expression only containing add, subtract and multiply operations, we get:* $\sigma^T_{exp} \leq \sigma^S_{exp}$.

Proof: Clearly, $\sigma^T_{a\circledast b}$ is tighter than the $\sigma^S_{a\circledast b}$ if two floating-point numbers a and b have no rounding error. If they are with rounding errors, we have $\sigma^S_a = B_a[(1+\varepsilon)^{k_a}-1]$ and $\sigma^S_b = B_b[(1+\varepsilon)^{k_b}-1]$, respectively. According to the conclusions drawn above, it can briefly be supposed that $\sigma^T_a \leq B_a[(1+\varepsilon)^{k_a}-1]$ and $\sigma^T_b \leq B_b[(1+\varepsilon)^{k_b}-1]$.

- $\sigma^T_{a+b} = \frac{|a\oplus b|\varepsilon}{1-\varepsilon} + \sigma^T_a + \sigma^T_b$ and $\sigma^S_{a+b} = (B_a+B_b)[(1+\varepsilon)^{max(k_a,k_b)+1}-1]$. So $\sigma^T_{a+b} \leq (B_a+B_b)\varepsilon + B_a[(1+\varepsilon)^{k_a}-1] + B_b[(1+\varepsilon)^{k_b}-1] \leq (B_a+B_b)[(1+\varepsilon)^{max(k_a,k_b)}+\varepsilon-1] \leq \sigma^S_{a+b}$. The derivation for $a-b$ is similar.
- $\sigma^T_{a\times b} = \frac{|a\otimes b|\varepsilon}{1-\varepsilon} + |b|\sigma^T_a + |a|\sigma^T_b + \sigma^T_a\sigma^T_b$ and $\sigma^S_{a\times b} = (B_a \times B_b)[(1+\varepsilon)^{k_a+k_b+1}-1]$. So $\sigma^T_{a\times b} \leq B_aB_b\varepsilon + B_bB_a[(1+\varepsilon)^{k_a}-1] + B_aB_b[(1+\varepsilon)^{k_b}-1] + B_aB_b[(1+\varepsilon)^{k_a}-1][(1+\varepsilon)^{k_b}-1] = B_aB_b[(1+\varepsilon)^{k_a+k_b}+\varepsilon-1] \leq \sigma^S_{a\times b}$.

According to mathematical induction, we can conclude that $\sigma^T_{exp} \leq \sigma^S_{exp}$ for a floating-point expression.