Accurate and Efficient Continuous Collision Detection

Wouter L. B. Buddingh'

***Abstract*—With continuous-collision-detection (or C.C.D.), instead of evaluating a scene statically (at certain moments in time), a scene is continuously evaluated (over a time interval). In this paper, a C.C.D. method is proposed based on feature-testing. Examples of features are points, edges and polygons. If we can test points and polygons, and edges and edges, we can perform C.C.D. on any polyhedral shape.**

Our method does not compromise on motion accuracy, instead our method delivers first-order-accuracy instead of zeroth-order-accuracy, which is the accuracy achieved by many other C.C.D. methods. We call our method D2M or differentiate-twice-method.

The constructs in this paper can be used for brute-force C.C.D. . For instance if someone would like to collide two polygons, or two cubes for instance. We will also show how to use a bounding-volume-hierarchy or BVH in combination with our method. By using this BVH, we can collide model-pairs in the order of 1 million triangles.

***Index Terms*—IEEEtran, journal, paper, continuous-collision-detection, feature-testing, bounding-volume-hierarchy.**

I. Introduction

Whenever a rigid-body simulation needs to be performed, such a simulation is usually accompanied by a discrete-time-collision-detection-system. Questions like; are two bodies intersecting? and where/how are the bodies penetrating each-other?, are both answered by the discrete-time system. With continuous-collision-detection (C.C.D.), any object is persistent in space while traveling and objects are not allowed to deform. In [13], this is referred to as truly-continuous-collision-detection. In general we are allowed to speak of truly-continuous-motion as motion is obtained by rigid-transformation. For a single object, it is possible to define a beginning state and an end state. The infinite amount of intermediate states are to be defined by the C.C.D. system.

Continuous-collision-detection is somewhat more difficult than discrete-time collision detection. Note that collision detection is clearly separated from collision response. Whereas the problem of collision detection is clearly postulated, this does not seem to be the case for collision response. Newton's impact law might lead to a unique solution for two colliding spheres. It is unclear how to apply this law to two colliding bunnies.

With C.C.D. the term TOC arises, TOC meaning *Time of Contact*. It is used to designate the first moment in time when generally only two objects are touching. When we restrict ourselves to two objects, this does not limit the concept to being used in a scene with just two objects. When many objects collide simultaneously, this can be re-factored in many two-body TOC queries. The number of two-body TOC-queries goes up quadratically as the number of simultaneously colliding objects increases.

Let $D \in \mathcal{P}(\mathbb{R}^3)$, stated differently let D be a point-set describing the externals and internals of an object. Now let $D(t)$ be D under some rigid transformation which is time (t) dependent. Likewise, let $A(t)$ and $B(t)$ both be time-dependent rigid-transformations of point-sets.

Static (discrete-time-) collision detection can be posed as:

$$\min_{t\in\mathbb{N}} (\text{collideStatic}(t)),$$
$$\text{where collideStatic}(t) = (A(t) \cap B(t) \neq \emptyset).$$

Likewise, continuous collision detection can be posed as:

$$\min_{t\in\{0\}\cup\mathbb{R}^+} (\text{collideStatic}(t))$$

Both definitions above assume that there is a moment of contact (TOC). The latter need not be the case because the possibility exists that there is no contact. In this case, the algorithm will return a positive infinite value. Within any practical application of C.C.D, the domain will be bounded instead of being $\{0\}\cup\mathbb{R}^+$. Multiple authors bound the domain to $[0, 1]$.

In [16], an alternate definition is used, considering one moving and one static object. This setup seems to be used for explanatory reasons. It must be noted that this explanation oversimplifies the problem. Without increasing the motion-complexity of the non-static object, two objects under rigid transformation cannot always be simplified to one moving and one static object, by merely changing the frame-of-reference.

The subclass of rigid transformation motions that is used we call: *Linear translational and linear rotational motion*. Intuitively this means that the centroid of an object is moving with linear motion. The object is simultaneously rotating around a fixed axis with a fixed angular-velocity.

This paper contributes to C.C.D. in the following ways:

- Quick, robust and highly accurate feature/feature tests:
  - Vertex/triangle
  - Edge/edge
- A new bounding-volume-hierarchy is introduced
- Two modes of operation of D2M
  - D2M (with Raw Continuous Collision Detection) designated with D2M
  - D2M (without Raw Continuous Collision Detection), just broad-phase, designated with D2M *Capoeira* (see Figures 1 and 4)
- Our accurate method (D2M) is very tight $[1, 3.3] \cdot 10^{-10}$ units separation distance
- The accuracy of D2M exceeds the accuracy of C2A (a competing system) by a big factor

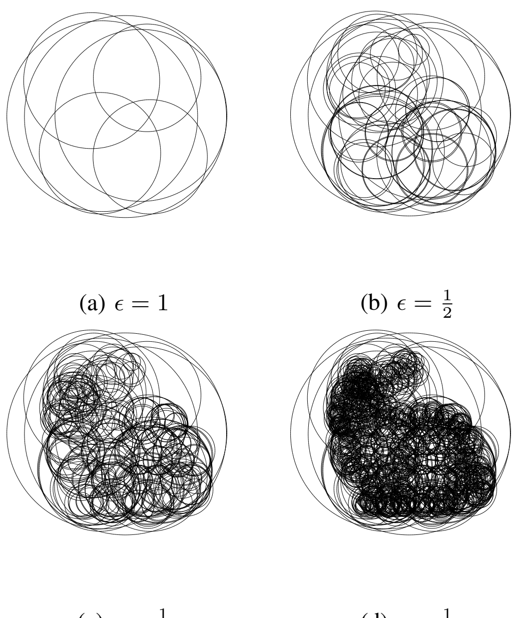
(a) $\epsilon = 1$ (b) $\epsilon = \frac{1}{2}$ (c) $\epsilon = \frac{1}{4}$ (d) $\epsilon = \frac{1}{8}$

Fig. 1: Sphere hierarchy with different epsilon levels. Within the benchmarks of this paper we use $\epsilon = \frac{1}{32}$

- Computational efficiency of D2M (Capoeira) rivals/exceeds the computational efficiency of C2A
- Able to do rigid-body simulation with C.C.D. and collision response (including a basic type of friction [2])
- Can be extended with any $\mathcal{C}_2$ continuous rigid-motion

In Section II, we refer to the works of others that are somehow related to this work. Section III, defines the criteria for the continuous in-between motion that can be used. Section IV deals with finding the first root of some arbitrary feature/feature test. In Section V, we describe what to do in order to assure disjointness after a collision. Section VI, provides a few hints on how to make the overall system perform faster. Section VII opens some room for discussion. Afterwards, in Section VIII, the testing results of D2M/D2M Capoeira, are compared to C2A. In Section IX, this paper is concluded. February 3, 2017

II. Related Work

Work has been done by Redon et al. [3]. to solve the C.C.D. problem in a neat way. Yet similar work was done in 1984 by Canny [1]. In the latter two papers, the problem is solved using complicated algebra. Both papers use a specific motion type, and both motion classes are subclasses of rigid motion. The motion class in this paper has preference over the other motion types. This motion type is comprehensible: "rotating with constant angular velocity while translating with constant velocity". The motion type used within this paper, can be regarded as natural. Instead of imagining a path in-between, a continuous path is chosen that is identical to the path obtained by an Euler integration step (by using a specific interpretation of Euler integration).

A while back in time, Mirtich [10], introduced the concept of conservative advancement, a technique for TOC querying on convex objects. Typically, conservative advancement relies on closest-point-pair queries. Such queries may be facilitated by e.g. the Lin-Canny- / Gilbert-Johnson-Keerthi- algorithm.

In general, objects are not convex. The latter is one reason why Min Tang et al. [16] devised the system "Controlled Conservative Advancement" or C2A. Within this system a proximity query library is used called PQP, which has been devised by Eric Larsen and Stefan Gottschalk. Next to supporting C2A, PQP also plays a crucial role in validating the results of the system presented in this paper.

In order to compare the performance of the system corresponding to this thesis, a state of the art competing method was chosen, namely C2A. Instead of relying on convex decomposition of generic polyhedral models (like C2A does), our system relies on primitive / primitive TOC queries, in combination with some hierarchical techniques.

A. Root-Finding

When a root cannot be found analytically or is sufficiently hard to find analytically, one often chooses a standard numerical method to begin with. One can choose a regula-falsi method if one is able to pinpoint two points $a, b \in \mathbb{R}$ such that $\text{sign}(f(a)) = -\text{sign}(f(b))$ and $a \neq b$. If one chooses this specific method, one can find the roots of fairly complicated functions. However, the user of this function, is still responsible for pinpointing a unique root within a certain domain.

In [6], a method as sketched above is used. The reason why such a method is not used within the context of this paper, is because it is particularly hard to pinpoint a domain in which the root has to occur. This argument holds for a lot of root-finders, they seem to be quite capable of finding a root within a given domain. Other root finders, like Newton-Raphson / the Householder's methods method are not guaranteed to find exceptional roots.

A few traits were important before our root-finder was devised:

- Robustness
- Absolute running time

Interval-arithmetic, as described in [14], seems to be obvious choice when one can spare the processing cycli. We have chosen to use a second-order inclusion-function which can also be called a "second-order Taylor-form" [15].

In [9] a lot is written about Taylor-models. A Taylor-model is a vertically thickened polynomial. Our inclusion-function is not a Taylor-model, as can be seen in Figure 2. Note that the upper and lower bounding parabolas are diverging. With Taylor-models, the vertical distance would be constant. Choosing the parabolas to be divergent, makes it a lot more difficult to apply basic arithmetic on the models, like is done in [19]. E.g. computing the product of two of such models would be a difficult task.

B. Bounding-Volume-Hierarchies

The concept of a Bounding Volume Hierarchy is required in order to get a decent processing rate. The main idea is:

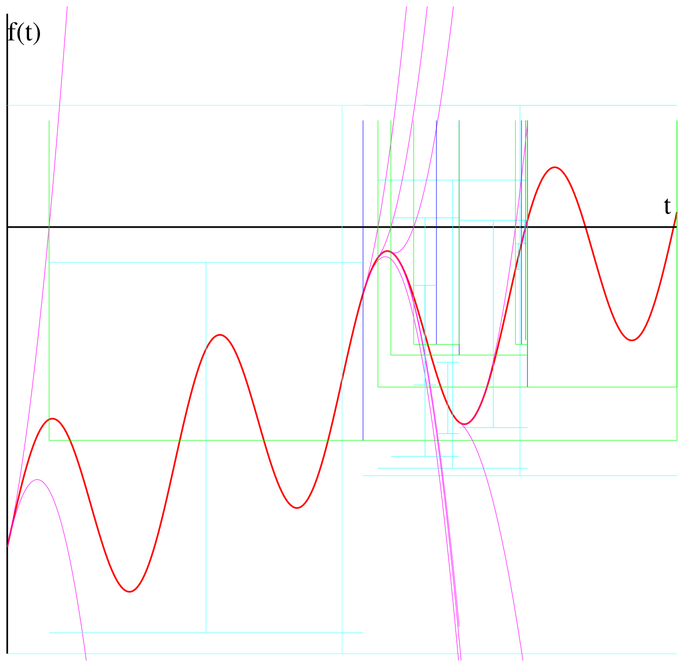

Fig. 2: Operation of the root finder. Inclusion-functions are shown in purple.

"Do not test what does not need to be tested.". We can draw a sphere around a very complicated object (A). If object B does not hit the sphere, there is no need to intersect with A. This idea can be exploited recursively, in order to get acceptable collision query times.

In [7], oriented bounding boxes are the basic building blocks of a Bounding Volume Hierarchy. The complexities involved with oriented bounding boxes, are in a different class than spheres and axis-aligned bounding boxes. In [18], van den Bergen measures that axis-aligned bounding-boxes cause more inner primitive tests than oriented bounding-boxes. This is because oriented bounding boxes, give a better fit. The latter seems to be of great importance.

Within the context of this paper, a sphere hierarchy is used. A property of these spheres is that they are smallest enclosing spheres. When dealing with hierarchies, it may be tempting to simply unite two spheres at the same level into a bigger, enclosing sphere at the parent-level. This strategy will result in a big sphere around the object in question, which is not what we prefer. In order to get an optimal hierarchy, the raw geometry at a certain level has to be fed to the minimal-enclosing-sphere-constructor-function. A sphere hierarchy associated with this project is constructed after a hierarchy of axis-aligned bounding-boxes or R-tree has been constructed. A lot is stated on R-trees in [8]. Results of [8] have been used within this paper. For instance, due to [8] we have chosen the number of children within the R-tree to be 2 at each inner-node, thereby excluding octree-like constructs.

III. Motion

The rigid in-between motion can conveniently be defined by a time-dependent 4×4 homogeneous matrix: $\mathbf{M}(t) = \begin{pmatrix} \mathbf{R}(t) & \boldsymbol{x}(t) \\ (0,0,0) & 1 \end{pmatrix}$ Where $\boldsymbol{x}(t)$ is the translational displacement, and $\mathbf{R}(t)$ is a right-handed orthonormal 3×3 matrix. Alternatively one can use a time-dependent pair of a quaternion and a 3-vector. What is important is that the definition is explicit. We either have a mapping from time to $\mathbb{R}^{4\times4}$, or a mapping from time to quaternion and 3-vector. When the motion mapping has been established, it needs to be "promoted" to interval-arithmetic. This way it is possible to see what happens in time-ranges instead at moments in time. In order to create these interval-mappings, one most-likely requires interval to interval-functions such as $\sin : \mathbb{IR} \mapsto \mathbb{IR}$, where $\mathbb{IR}$ denotes the set of intervals.

IV. Computing TOC for feature/feature tests

It is very difficult to solve the continuous-collision-detection problem as a whole. Thankfully, the problem permits itself to be subdivided. Let the problem in its bare form be TOC determination. It is enough to be able to determine the TOC between two moving features e.g. a vertex and a convex polygon. This TOC computation together with edge/edge TOC determination, is sufficient to compute the TOC between two non-convex bodies under some rigid motion.

Whether it concerns point/polygon tests or edge/edge tests, we require a root finder that finds the first root of a complicated function. Also, while searching for roots we would like to exclude parts of the search domain that cannot produce a viable solution. The function $f : \mathbb{R} \mapsto \mathbb{R}$ is required to be in continuity-class $\mathcal{C}_0$ for Algorithm 1 and in continuity-class $\mathcal{C}_2$ for Algorithm 3.

The definition of f may be very complicated. We can think of f as a distance function. The distance between two features for instance. Whenever the distance reaches 0, there is a collision. Preferably this distance function can also be negative.

First let us start with the plain interval-arithmetic version of the solver. See Algorithm 1.

```
Algorithm: $BBOI$
input : An interval function: $f : \mathbb{IR} \mapsto \mathbb{IR}$
        A boolean function: $\Gamma : \mathbb{IR} \mapsto \mathbb{B}$
        A domain in which to search: $[t_0, t_1] \in \mathbb{IR}$
        A time based epsilon value: $\epsilon_t \in \mathbb{R}$
output: A root is (possibly) declared $[r_0, r_1] \in \mathbb{IR} \cup \emptyset$
1 if $\Gamma([t_0, t_1]) \wedge (0 \in f([t_0, t_1]))$ then
2     if $t_1 - t_0 \leq \epsilon_t$ then return $[t_0, t_1]$;
3     middle $\leftarrow \frac{t_0+t_1}{2}$;
4     root $\leftarrow BBIO(f, \Gamma, [t_0, \mathrm{middle}], \epsilon_t)$;
5     if root $\neq \emptyset$ then return root;
6     return $BBOI(f, \Gamma, [\mathrm{middle}, t_1], \epsilon_t)$;
7 return $\emptyset$;
```

Algorithm 1: Binary-Bisection-On-Intervals, a recursive algorithm that can be used to find roots with arbitrary precision (see ϵ_t). Note that the algorithm does not account for floating point issues. E.g. $\frac{t_0+t_1}{2}$ could evaluate to t_0.

The observant reader may have noticed the Γ function. This function can be used to efficiently validate the output. If the input of the Γ function could contain a valid root (i.e. an intersection), then the function must return **true** . Each root of f does not imply the presence of an intersection. It is much easier to simplify the "distance" function f such that it may report false roots. The false roots are later rejected by our boolean function Γ.

For instance, formulating a positive distance-function between a polygon and a vertex is not only quite hard, the root-finding process is also likely to be less robust.

Within the context of this project, we basically compute collisions of vertices and planes and prune the ones that do not hit our specified polygon. Thanks to our function Γ, the latter two processes can be done simultaneously.

A. Accelerating root-finding

We can accelerate the computation of roots, by using some derived features of f, let us say, the the value of function f evaluated over a time-domain: $[f_0, f_1]$ (using interval-arithmetic), the time-derivative of f, f' and the second time-derivative of f evaluated over a time-domain $[f''_0, f''_1]$ (using interval-arithmetic).

```
Algorithm: $HRFOI$
input : A real function: $f : \mathbb{R} \mapsto \mathbb{R}$
        An interval function (promoted): $f : \mathbb{IR} \mapsto \mathbb{IR}$
        Its time derivative $f' : \mathbb{R} \mapsto \mathbb{R}$
        The second time derivative $f'' : \mathbb{IR} \mapsto \mathbb{IR}$
        A boolean function: $\Gamma : \mathbb{IR} \mapsto \mathbb{B}$
        A domain in which to search: $[t_0, t_1] \in \mathbb{IR}$
        A time based epsilon value: $\epsilon_t \in \mathbb{R}$
output: A root is (possibly) declared $[r_0, r_1] \in \mathbb{IR} \cup \emptyset$
1  if $\neg\Gamma([t_0, t_1]) \vee 0 \notin f([t_0, t_1])$ then return $\emptyset$;
2  $g \leftarrow f(t_0)$ ;
3  $g' \leftarrow f'(t_0)$ ;
4  $[g''_0, g''_1] \leftarrow f''([t_0, t_1])$ ;
5  $I \leftarrow \theta(g, g', \frac{[g''_0, g''_1]}{2}, [t_0, t_1])$ ;
6  $[h_0, h_1] \leftarrow IRD(I, [t_0, t_1])$;
7  if $h = \emptyset$ then return $\emptyset$;
8  if $h_1 - h_0 \le \epsilon_t$ then
9      if $\Gamma([h_0, h_1])$ then return $[h_0, h_1]$;
10     return $\emptyset$;
11 middle $\leftarrow \frac{h_0 + h_1}{2}$;
12 root $\leftarrow HRFOI(f, f', f'', \Gamma, [h_0, \text{middle}], \epsilon_t)$ ;
13 if root $\neq \emptyset$ then return root;
14 return $HRFOI(f, f', f'', \Gamma, [\text{middle}, h_1], \epsilon_t)$ ;
```
Algorithm 2: Hybrid-Root-Finder-On-Intervals.

The inclusion-function is characterized by $ax^2 + bx + c$, where $a \in \mathbb{IR}$ and $b, c \in \mathbb{R}$. The extremes of a define the upper-/lower-parabola.

$$\theta(f, f', [f''_0, f''_1], [t_0, t_1]) = [f + f'x + \tfrac{1}{2}f''_0 x^2, f + f'x + \tfrac{1}{2}f''_1 x^2].$$

Where $x = t - t_0$, and t represents time (see Figure 2). Note that $\theta_0 \le \theta_1$. Function $computeFirstUpperRoot : \mathbb{IR} \mapsto \mathbb{R} \cup \infty$, computes the first root of the upper-parabola of an inclusion-function within the given domain. Likewise $computeFirstLowerRoot : \mathbb{IR} \mapsto \mathbb{R} \cup \infty$, computes the first root of the lower-parabola of an inclusion function within the given domain. If no such root exists, both functions yield ∞.

```
Algorithm: $IRD$
input : An inclusion function: $I$
        A domain which to shorten: $[t_0, t_1] \in \mathbb{IR}$
output: The shortened search domain (optional)
        $[r_0, r_1] \in \mathbb{IR} \cup \emptyset$
1 if $I(t_0) < 0$ then $I \leftarrow -I$;
2 $l \leftarrow I.computeFirstLowerRoot([t_0, t_1])$;
3 if $l \notin \mathbb{R}$ then return $\emptyset$;
4 $u \leftarrow I.computeFirstUpperRoot([t_0, t_1])$;
5 if $u \notin \mathbb{R}$ then return $[l, t_1]$;
6 return $[l, u]$
```
Algorithm 3: Intersect-Root-Domain. This function will compute the root-domain, and intersect it with the given domain.

If the second-derivative interval of inclusion function I is $[0, 0]$, then the inclusion function represents a line. The slope of the inclusion function can be 0 too. In case of a horizontal line, it could be the case that the interception of the line is also 0, in which case we have infinitely many roots to declare. This singularity is dealt with by simply discarding horizontal line-segments. For the primitive-primitive tests done within this paper, discarding horizontal-line segments is the proper action.

B. Point / Triangle test

Up until now, we have used the term convex-polygon instead of triangle. Using triangles instead of convex-polygons within computational-geometry has the advantage that a triangle is only invalid when two vertices are equal. When using convex-polygons, one is troubled with checking whether the polygons are really flat and convex.

We will ask ourselves the question: "Will a point following a particular trajectory, collide with a moving triangle and when will this event occur?". In Section III, we already described the type of motion. So both the vertex and the triangle have to adhere to such a motion type i.e. rigid-motion. There are two objects, object A and object B, each having a well defined rigid transformation dependent on time $\mathbf{A}(t)$ and $\mathbf{B}(t)$ respectively. The point $\boldsymbol{a}$ is static in the local space of object A, and is transformed by $\mathbf{A}(t)$ to form $\boldsymbol{a}(t)$ in world-space. The points $\boldsymbol{b}$ and $\boldsymbol{c}$ are static in the local space of B and transformed by $\mathbf{B}(t)$ to form $\boldsymbol{b}(t)$ and $\boldsymbol{c}(t)$ in world-space. Let $\boldsymbol{b}$ be a point on our polygon. The polygon has a normal $\hat{\boldsymbol{n}}$, which is defined to be $\boldsymbol{c} - \boldsymbol{b}$. In world-space, let us define $\hat{\boldsymbol{n}}(t) = \boldsymbol{c}(t) - \boldsymbol{b}(t)$. Let us also define $\boldsymbol{\gamma}(t) = \boldsymbol{a}(t) - \boldsymbol{b}(t)$. Now we can define a function that happens to be the signed-distance of point $\boldsymbol{a}(t)$ to the plane supporting our polygon:

$$f(t) = \boldsymbol{\gamma}(t) \cdot \hat{\boldsymbol{n}}(t).$$

Recall that our accelerated root finder required not only f, but f, f' and f''. The first and second analytical time-derivatives of a point undergoing linear translational and linear rotational motion are given in Appendix A. The time-derivatives of $f(t)$ are derived in Appendix D.

```
Algorithm: $\Gamma_{PT}$
input : A vertex: $\boldsymbol{a} \in \mathbb{IR}^3$ defined in B-space.
        A triangle: $\boldsymbol{T}_i \in \mathbb{R}^3$, where $i \in \{1,2,3\}$
defined in B-space.
output: A boolean indicating whether an intersection
        could occur.
1 $\mathbf{M} \leftarrow$
    $\big(\boldsymbol{T}_2 - \boldsymbol{T}_1 \quad \boldsymbol{T}_3 - \boldsymbol{T}_1 \quad (\boldsymbol{T}_2 - \boldsymbol{T}_1) \times (\boldsymbol{T}_3 - \boldsymbol{T}_1)\big)^{-1}$;
2 $\boldsymbol{x} \leftarrow \mathbf{M}(\boldsymbol{a} - T_1)$;
3 if $x_{x,1} < 0 \vee x_{y,1} < 0 \vee x_{x,0} + x_{y,0} > 1$ then
4 |   return false ;
5 return true ;
```

Algorithm 4: Γ-Point-Triangle. Before using this algorithm, the point $\boldsymbol{a}$ must be converted to B-space. Because the gamma function is given a time-span instead of a time-moment, $\boldsymbol{a}$ becomes a 3D-spatial interval. This function preforms a 2D-barycentric triangle test. The matrix computation at line 1 has to be done once for each triangle, and can thus be optimized away. Because the third term of $\boldsymbol{x}$, x_z is not used, the vector/matrix multiplication in line 2 can be replaced with two dot products.

Note that instead of 4×4 matrices, 3-vector/quaternion pairs are preferred in order to represent a transformation. The latter makes the math more complicated but will result in better computational performance.

C. Edge / Edge test

Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be transformed into world-space by $\mathbf{A}(t)$ into $\boldsymbol{a}(t)$ and $\boldsymbol{b}(t)$. In a similar way, let $\boldsymbol{c}$ and $\boldsymbol{d}$ be transformed into world-space by $\mathbf{B}(t)$ into $\boldsymbol{c}(t)$ and $\boldsymbol{d}(t)$. Next, let $\boldsymbol{\delta_1}(t)$ be defined as $\boldsymbol{b}(t) - \boldsymbol{a}(t)$, and let $\boldsymbol{\delta_2}(t)$ be defined as $\boldsymbol{d}(t) - \boldsymbol{c}(t)$. Next, let $\boldsymbol{n}(t) = \boldsymbol{\delta_1}(t) \times \boldsymbol{\delta_2}(t)$ be the direction in which the distance between the two supporting lines should be measured. If the lines are parallel, $\boldsymbol{n}(t) = \mathbf{0}$. In the case of non-parallel lines, our distance function $f(t)$ equals $s(t)\boldsymbol{n}(t) \cdot (\boldsymbol{c}(t) - \boldsymbol{a}(t))$, where $s(t) = \frac{1}{\|\boldsymbol{n}(t)\|_2}$. If we drop the normalization of $f(t)$, its roots will not be altered. Let $f(t)$ be $\boldsymbol{n}(t) \cdot (\boldsymbol{c}(t) - \boldsymbol{a}(t))$. The time-derivatives of $f(t)$ are given in Appendix C.

Next, we can pose the question; "How would $\Gamma_{EE} : \mathbb{IR} \mapsto \mathbb{B}$ look like?". For each edge, we have two vertices dependent on a time-interval. With interval-arithmetic, these vertices are represented as axis-aligned bounding boxes.

Each of these bounding boxes has a width (w), a height (h) and a depth (d). Using the Pythagorean theorem it is possible to compute a radius for a bounding sphere: $\sqrt{\frac{w^2+h^2+d^2}{4}}$. An edge can be seen as the convex-hull spawned by two of these bounding spheres. The Γ_{EE} function is only used to prune roots. Because we are dealing with roots, we may assume the lines extending the edges are intersecting. The problem thus exists within a plane spawned by the two intersecting lines. We can thus make a 2D-projection of the 3D-edge-edge scene. In case the capsules are non-parallel, the axis of the plane is defined by $(\boldsymbol{b'} - \boldsymbol{a'}) \times (\boldsymbol{d'} - \boldsymbol{c'})$, where the primed-vectors are the origins of the associated spheres. If the edges are parallel, the axis is defined by: $(\boldsymbol{c'} - \boldsymbol{a'}) \times (\boldsymbol{d'} - \boldsymbol{a'})$. See figure 3

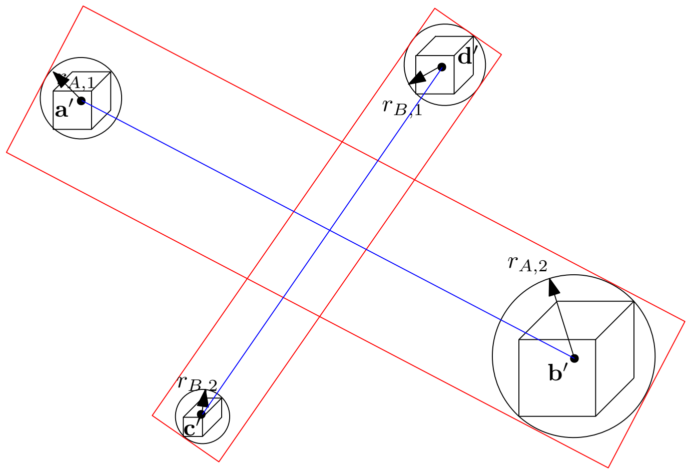

Fig. 3: Two 3D-capsules are projected on the plane. The cross product of the length axis of both capsules is orthogonal to the projection plane.

The eventual test is just a rectangle/rectangle-intersection test. A rectangle/rectangle test is conceptually identical to the convex polygon/convex polygon test described in [5].

D. Sphere/sphere test

As mentioned before, we can use spheres as bounding volumes. Because we are in possession of a minimal-enclosing-sphere algorithm [4] [11], instead of colliding the geometry inside a sphere, we can test two bounding spheres for intersection first.

Let there be two spheres, sphere A and sphere B. Let $\boldsymbol{a}(t) \in \mathbb{R}^3$ be the origin of sphere A and let $\boldsymbol{b}(t) \in \mathbb{R}^3$ be the origin of sphere B. The radii of the two spheres are denoted by r_A and r_B.

When the distance between the origins is equal to the sum of the radii, then surfaces of the spheres are touching each-other. To be explicit, let $\boldsymbol{\gamma}(t)$ be $\boldsymbol{b}(t) - \boldsymbol{a}(t)$. If $\|\boldsymbol{\gamma}(t)\|_2 = r_A + r_B$ then the surfaces of the spheres are touching.

Because $\|\boldsymbol{\gamma}(t)\|_2$ is defined as $\sqrt{\boldsymbol{\gamma}(t) \cdot \boldsymbol{\gamma}(t)}$, the derivatives are somewhat harder to derive. However, by squaring the formula we do get something that is usable:

$$\|\boldsymbol{\gamma}(t)\|_2^2 = (r_A + r_B)^2$$
$$\|\boldsymbol{\gamma}(t)\|_2^2 - (r_A + r_B)^2 = 0.$$

Let us define the expression above as $f(t)$. When $f(t) = 0$, the surfaces of the spheres are touching. We need to find (a lower bound of) the first occurrence of t such that $f(t) = 0$:

$$f(t) = \boldsymbol{\gamma}(t) \cdot \boldsymbol{\gamma}(t) - (r_A + r_B)^2. \qquad (1)$$

The time-derivatives of $f(t)$ are listed in Appendix B.

During benchmarking, it has been shown that the computational cost of evaluating f, $[f_0, f_1]$, f' and $[f''_0, f''_1]$ is high. To minimize the number of evaluations, the number of iterations used is just one. In order to maximize bounding quality of the inclusion function, the time domain is shortened. Collisions within D2M are computed over an arbitrary time-domain. Some authors always use the time-domain $[0, 1]$. By subdividing the time-domain of the entire-collision even further, the bounding quality of our parabolas increases. The latter may cause scaling the number of subdivisions of our time-domain to have an unexpected positive effect on the total computation time of the collision.

Thus, by decreasing the time-span, and thus searching a domain in more steps, the performance is actually increased, because the bounding quality of each segment is sufficiently increased. However, there does seem to be some optimum between time-span and bounding-quality.

A sphere/sphere test using a single iteration is a continuous broad-phase collision detection primitive. A swept-volume axis-aligned bounding-box is computed using interval arithmetic for each sphere. This bounding-box, together with single iteration data is stored for each sphere. The swept-volume bounding-boxes supply quick but inaccurate non-intersection tests. If they indicate possible intersection, then a quick single iteration sphere/sphere test is done. The sphere/sphere tests are not the bottleneck. The computation of sphere associated data is.

With sphere associated data, we mean the origin of a sphere, at the beginning of the trajectory as well as the interval-origin during the entire trajectory.

The following properties of a Sphere-Over-Domain object are maintained.

- $x_{t,0}$, the origin of the sphere at $t = 0$
- $x_{t,\text{domain}}$, wherever the origin can be during the time-domain
- $\dot{x}_{t,0}$, the velocity of the sphere at $t = 0$
- $\dot{x}_{t,\text{domain}}$, any velocity the sphere can reach during the time-domain
- $\ddot{x}_{t,\text{domain}}$, any acceleration the sphere can reach during the time-domain
- r, the radius of the sphere
- BB, a bounding-box bounding the space where the sphere can go during the time-domain

In Algorithm 5, a domain of intersection is computed. This domain can be the empty set ($\emptyset$, i.e. no collision).

```
Algorithm: $TSP$
input : An interval: $[t_0, t_1] \in \mathbb{IR}$
        $A_{SOD}$, $B_{SOD}$
output: An interval describing an overlap period
        (optional):$[r_0, r_1] \in \mathbb{IR} \cup \emptyset$
1  if $\neg A_{SOD}.BB.intersects(B_{SOD}.BB)$ then return $\emptyset$;
2  $\gamma_{t,0} \leftarrow B_{SOD}.\boldsymbol{x}_{t,0} - A_{SOD}.\boldsymbol{x}_{t,0}$;
3  $\dot{\gamma}_{t,0} \leftarrow B_{SOD}.\dot{\boldsymbol{x}}_{t,0} - A_{SOD}.\dot{\boldsymbol{x}}_{t,0}$;
4  $\gamma_{t,\text{domain}} \leftarrow B_{SOD}.\boldsymbol{x}_{t,\text{domain}} - A_{SOD}.\boldsymbol{x}_{t,\text{domain}}$;
5  $\dot{\gamma}_{t,\text{domain}} \leftarrow B_{SOD}.\dot{\boldsymbol{x}}_{t,\text{domain}} - A_{SOD}.\dot{\boldsymbol{x}}_{t,\text{domain}}$;
6  $\ddot{\gamma}_{t,\text{domain}} \leftarrow B_{SOD}.\ddot{\boldsymbol{x}}_{t,\text{domain}} - A_{SOD}.\ddot{\boldsymbol{x}}_{t,\text{domain}}$;
7  $s_{t,0} \leftarrow \|\gamma_{t,0}\|_2^2 - (A_{SOD}.r + B_{SOD}.r)^2$;
8  $\dot{s}_{t,0} \leftarrow 2(\gamma_{t,0} \cdot \dot{\gamma}_{t,0})$;
9  $\ddot{s}_{t,domain} \leftarrow 2(\ddot{\gamma}_{t,\text{domain}} \cdot \gamma_{t,\text{domain}} + \|\dot{\gamma}_{t,\text{domain}}\|_2^2)$;
10 $I \leftarrow \theta(s_{t,0}, \dot{s}_{t,0}, \frac{\ddot{s}_{t,domain}}{2}, [t_0, t_1])$ ;
11 $l, u \leftarrow I.computeLowerRoots([t_0, t_1])$;
12 $r \leftarrow [t_0, t_1]$;
13 if $s_{t,0} > 0$ then
14     if $l \in \mathbb{R}$ then
15         $r \leftarrow r \cap [l, \infty]$;
16         if $u \in \mathbb{R}$ then $r \leftarrow r \cap [-\infty, u]$;
17     else
18         return $\emptyset$;
19 else
20     if $l \in \mathbb{R} \wedge u \notin \mathbb{R}$ then $r \leftarrow r \cap [-\infty, l]$;
21 return $r$;
```

Algorithm 5: Test-Sphere-Pair. The function computes the time-overlap-interval of two moving spheres. The interval is conservative, e.g. the case $s_{t,0} \leq 0$ could yield two intervals. In such a case, an intermediate non-intersection is reported as intersection. In line 10 the inclusion function is constructed. The math above line 10 is derived in Appendix B. Function *computeLowerRoots* : $\mathbb{IR} \mapsto (\mathbb{R} \cup \infty)^2$ operates on an inclusion-function and computes the roots of the lower-parabola within the given domain. The roots are returned in sorted order.

V. Ensure disjointness, take a step back

A problem with these kind of systems is that the objects are not disjoint at the moment of collision. The earliest interval that gets reported represents a touching collision, which is not what we want. If we pick the beginning of the interval returned, we still do not get a collision where both objects are strictly disjoint. The latter is due to arithmetical error. Systems like C2A, do not have this problem. Because the C2A method relies on distance queries, it is possible to ensure that a minimal distance is maintained between the objects.

An unrefined method to get disjoint objects with high probability is to decrement the TOC with some small constant, and then clamp the value to the non-negative domain. When using this construct, as the collision speeds increases, so does the distance between the objects at the time of collision. This effect is unwanted, and can easily be compensated. At the time of the raw collision, a contact-normal exists pointing from object A to object B. We can measure the impact velocity along the contact normal ($v_{\hat{\boldsymbol{n}}}$). Additionally, we can specify a distance epsilon (ϵ_d), which represents the amount we wish the objects to be disjoint. We get the following equation $\Delta TOC \cdot v_{\hat{\boldsymbol{n}}} = \epsilon_d$. We can rewrite this into $\Delta TOC = \frac{\epsilon_d}{v_{\hat{\boldsymbol{n}}}}$. Hence we have to decrement the TOC with ΔTOC and clamp it to the non-negative domain.

The construct above works well when ϵ_d is set to the same order as the precision of the root finder. Note that the root finder precision is described in time units (ϵ_t). When the distance epsilon of is set to 10^{-10}, the distance that is reported by the PQP library is in between $1.0 \times 10^{-10} \sim 3.3 \times 10^{-10}$.

VI. Optimizing the system

Within the context of this project, a bounding-volume-hierarchy has been used. This bounding-volume-hierarchy is briefly described in Section VI-A. For any simulation system containing many objects to be useful, one needs a method to counter-act the quadratic complexity that arises between multiple objects. In [17], one can find variations of the Sweep and Prune algorithm, which reduces the computational complexity of a many-body-simulation.

What has not been solved with satisfaction is resting contact. Within C2A, a heuristic is used to avoid recomputing similar collisions. A similar construct can be used with D2M (non-capoeira) to reduce computations. Whenever two objects are colliding, it is likely that they will collide again at similar spots. By caching the most recently collided leaf-leaf pair of an object pair that collided, the next collision query can be accelerated.

By merely computing the TOC of the geometry within the two cached leaves, if there is a collision, between these subsets of geometry we can infer that there must be a collision between both objects in full.

In the case of a collision of subsets of the geometry, we only have to search the time-domain before the occurrence of such a collision. This will most likely reduce the total computation time.

If the subsets of the geometry did not collide, we have to test the geometry in full. In this case, there is a slight computational overhead.

The above heuristic only works well for D2M (non-capoeira). In capoeira-mode, there seem to be anomalies that conflict with this approach. Strictly, the geometry in capoeira-mode does not converge to the actual shape of the object. Therefore it is also debatable if capoeira-mode can be called “truly continuous”.

A. Our BVH method

An example of our bounding-volume-hierarchy class is shown in Figure 4. The class can be called an augmented R-tree.

The tree is built recursively by choosing an axis-aligned plane to subdivide a beam volume. The subdivided beam-volume contains two parts of geometry, either empty or non-empty. A minimal axis-aligned bounding-box is fitted around the geometry in case the geometry exists and subdivision is done again until a rib of the beam reaches a certain minimum length (ϵ).

For the subdivision-plane, we simply pick the broadest dimension of the beam and select the middle.

A C.C.D. collision query function takes two of these bounding-volume-hierarchies (A_{node} and B_{node}), see Algorithm 6.

The function $\mathit{TestSpherePair}(p, m_1, m_2)$ within Algorithm 6 tests two spheres having motions (m_1, m_2) for intersection according to Section IV-D. Note that the parameters of this function are different from the parameters in Section IV-D. These parameters can easily be converted.

Algorithm 6 has some floating-point arithmetical precision issues. These can be overcome in practice by keeping the floating-point numbers as close to zero as possible. At the start of the function, we can define $\texttt{epoch} \leftarrow t_0$. We can integrate the motions A_{motion} and B_{motion} in time such that their $t = 0$ also corresponds to $\texttt{epoch}$. Also, we can determine an approximate centroid of the scene at the beginning of the function and translate the entire scene such that the magnitudes of our numbers become smaller. Afterwards, if we decide to return contact-point data, we have to translate back again.

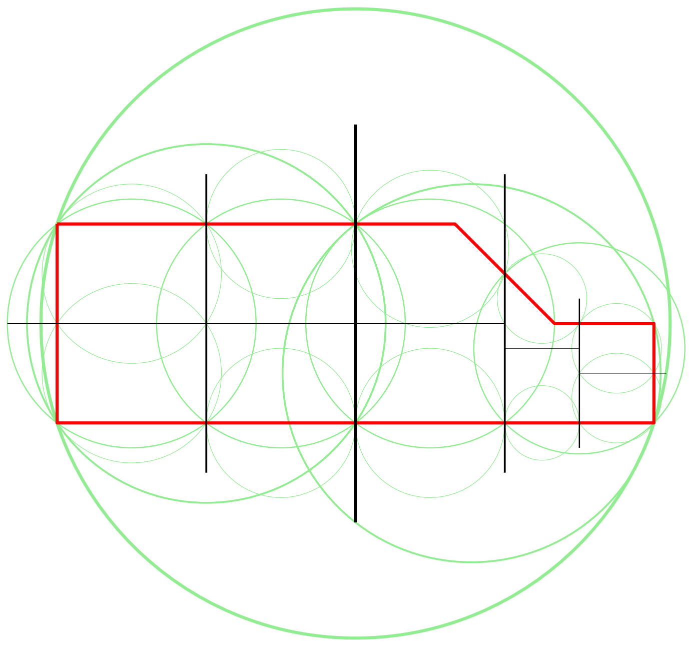

Fig. 4: A bounding-volume-hierarchy of a 2D car-like object. Note that the the subdivision process creates small beams (rectangles in the picture). The geometry within these beams or rectangles is used to construct minimum enclosing disks/-spheres. This process yields a very efficient BVH.

VII. Discussion

- For simulation purposes, when a collision has occurred, one may wish to alter the rotational and translational velocities of the objects. The latter can be done by applying bi-directional impulses on the contact points that are approximately shared by two objects.
  - We use a flawed method based on the Projected Gauss-Seidel method to accomplish this.
  - In case of severe dis-function, causing penetration, a hack is applied to prevent this.
  - Using impulse reactions is hard, because the contact times can be distributed in an insane manner.
  - Our simulator allows for at most 128 collision-events per frame (at 64 FPS). If more collision-events occur, the simulation time is skewed. A speed of 64 FPS is usually maintained in Capoeira mode. Even when three objects collide simultaneously (in resting contact), the frame-rate is still acceptable.
  - Resting contact could result in a low frame-rate.
- Next to impulse reactions, reaction forces can also be applied to make it realistic.
  - A method called Iterative Dynamics by Erin Catto, has partially been used to accomplish this. Originally, the method of Catto is used for permeable Rigid bodies. Impulses do not exist, because everything is handled by forces. The main thing to gain by using Catto’s method is normal-force that counter-acts, gravity and external force.
- Performance
  - A collision must be resolved by a single thread.

```
Algorithm: $Traverse - Tree - Pair$
input  : An interval: $[t_0, t_1] \in \mathbb{IR}$
         A node of each tree: $A_{\text{node}}$, $B_{\text{node}}$
         The motions of each object: $A_{\text{motion}}$, $B_{\text{motion}}$
output: $TOC \in [t_0, t_1] \cup \infty$
1  pairList ← [ ];
2  if $IsLeaf(A_{\text{node}}) \wedge IsLeaf(B_{\text{node}})$ then
3      /* Brute force the collision-geometry of $A_{\text{node}}$ and
          $B_{\text{node}}$. Return the TOC or $\infty$. In
          Capoeira mode, we return $t_0$.   */
4      return the above ;
5  else
6      for all parts $a$ of $A_{\text{node}}$ do
7          for all parts $b$ of $B_{\text{node}}$ do
8              $Add$(pairList, $(a, b)$) ;
9      /* N.B. In case a node is a leaf,
          it has only one part namely
          itself. Otherwise, its parts are
          its children.                 */
10 modifiedPairList ← [ ] ;
11 foreach $p$ in $pairList$ do
12     test ← $TestSpherePair$(p, $A_{\text{motion}}$, $B_{\text{motion}}$) ;
13     if $test \neq \emptyset$ then
        $Add$(modifiedPairList, ($test$, p));
14 /* Sort on lower time bound
      (ascending).                      */
15 $Sort$(modifiedPairList);
16 $TOC \leftarrow \infty$;
17 $TOC_1 \leftarrow t_1$;
18 foreach $modifiedPair$ in $modifiedPairList$
    do
19     $TOC_1 \leftarrow \min\{TOC, TOC_1\}$;
20     $TOC_2 \leftarrow \min\{TOC_1, \text{modifiedPair}_{1,1}\}$;
21     if $modifiedPair_{1,0} \leq TOC_2$ then
22         $TOC_3 \leftarrow Traverse - Tree - Pair$
23         ( [modifiedPair$_{1,0}$, $TOC_2$],
24           modifiedPair$_{2,1}$, modifiedPair$_{2,2}$,
25           $A_{\text{motion}}$, $B_{\text{motion}}$);
26         $TOC \leftarrow \min\{TOC, TOC_3\}$;
27 return $TOC$;
```

Algorithm 6: Query the TOC.

- Distributing the C.C.D. problem over multiple threads fails using either p-threads or OpenMP, CUDA® and OpenCL have not been attempted.
- All results are processed by a single thread of an Intel® Core™ i7-6700K CPU @ 4.00GHz × 8.

VIII. Results

Exhaustive benchmarks have been performed on pairs of objects (see Figure 5). Each set of models sized 2 is tested in 4 different ways: D2M versus C2A low speed (4 units per second), D2M Capoeira versus C2A low speed, D2M versus C2A high speed (512 units per second) and D2M Capoeira versus C2A high speed. Note that with this construct, C2A benchmarks are done twice. The double data-set serves as a sanity check, measuring timings twice should not cause too much deviation. After discarding one of the C2A datasets, each method is tested 5120 times ((speeds = 2) × (experiments = 10) × (pseudo-random-trials = 256)).

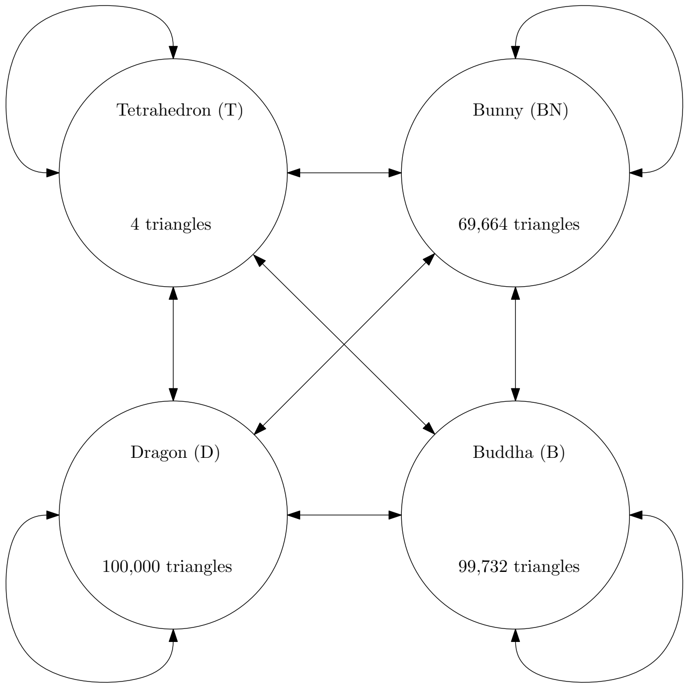

Fig. 5: Experiment Setup (10 combinations)

In Figure 6, an aggregation of all the distances at the collision instance for each benchmark is shown. Note that there are 40 benchmarks. Each benchmark is either D2M versus C2A or D2M Capoeira verus C2A.

The bar in the middle of the box-plot indicates the median. The dots indicate the mean values. The box extents indicate the lower/upper- quartile, whereas the upper and lower whisker indicate the minima and maxima.

Both diagrams have a logarithmic scale. The precision of D2M is quite exceptional. The non-penetration property of C2A does not always hold according to the second box and whisker plot. C2A fails in 9 of the 5120 test cases. The accuracy of D2M Capoeira is the poorest.

With D2M you get a contact distance of $1.0 \times 10^{-10} \sim 3.3 \times 10^{-10}$, whereas the distance obtained with C2A is within the range $0 \sim 5.1 \times 10^{-2}$. It can be stated that the quality of computation of D2M is better than the computation quality of C2A. A property of C2A, is that when the motion span increases, the minimal separation distance at the collision also increases. In [16], they seem to be using a fixed amount of effort in order to compute a single collision. In comparison, D2M takes as much processing power it takes in order to achieve its goal of "sub-nanometer" precision.

When collisions are scaled in such a way that the accuracy of D2M Capoeira matches that of C2A, i.e. when the relative velocity between the objects is 512 units per second, D2M Capoeria requires the least computational effort. This suggests that D2M Capoeira is well suited for low-accuracy, high-speed

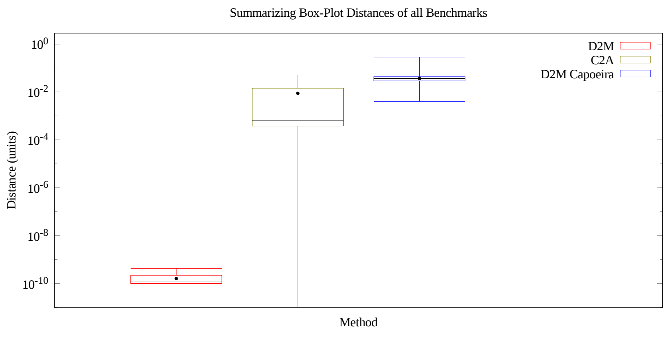

Fig. 6: Overall aggregated distances. Note that the distances of D2M (red) are within sub-nanometer precision (when considering the unit type to be meter).

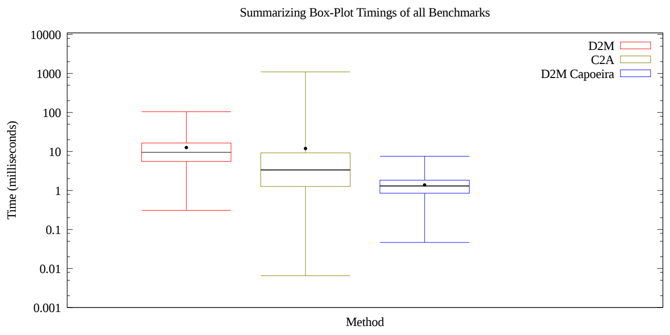

Fig. 7: Overall aggregated timings.

continuous-collision-detection. For more statistics on D2M, C2A and D2M Capoeira, see Appendix E.

IX. Conclusion

D2M is very precise, D2M Capoeria is imprecise but requires little computational effort. The C2A method is approached from two sides, D2M wins in precision, and D2M Capoeira wins when considering computational effort. With D2M or D2M Capoeira, it is possible to construct an impulse based simulation, possibly augmented with the force model found in [2]. We succeeded in doing so, however discussing continuous simulation is outside the scope of this paper.

Our system seems to be suited for real-time interactive simulation. However, we would advise caution to integrate D2M into time-critical applications. Our current simulation implementation sometimes suffers from in-acceptable time lag. You do not want your game-, simulation-, or (autonomous-)vehicle- system to wait for a certain amount of time. For any serious, time-critical applications, a fall-back mechanism is required.

X. Future Work

By considering early works in continuous simulation and re-evaluating them, there may be something to gain in this field. Our current simulator waists computation cycles simply by enforcing resting contact. A smart system would identify resting contact and enforce it without trying to resolve tiny bounces between models. The way the computations are optimized now, is non ideal for certain types of simulations. Our system heavily relies on interval-arithmetic. The C.C.D. problem is solved in [1] without the usage of interval-arithmetic (with only a slight modification to the motion-definition). Because (our implementation of) interval-arithmetic relies on conditional-jumps and because conditional-jumps are handled in a non-optimal manner by modern processors, our system has a disadvantage. It might be a good idea to construct another C.C.D. system based upon [1].

Appendix A
The 3D case

$$\dot{\boldsymbol{p}}(t) = \boldsymbol{\omega} \times \boldsymbol{r}(t) + \boldsymbol{v}. \tag{2}$$

The first derivative of a point is basically known (see Equation 2). The second derivative can be found by using the following identity:

$$\frac{d}{dt}(\boldsymbol{a} \times \boldsymbol{b}) = \dot{\boldsymbol{a}} \times \boldsymbol{b} + \boldsymbol{a} \times \dot{\boldsymbol{b}}. \tag{3}$$

One may observe that certain terms vanish. This has to do with the assumption that the motion is of the type “linear translational and linear rotational motion”. Applying $\frac{d}{dt}$ on equation 2, yields:

$$\ddot{\boldsymbol{p}}(t) = \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}(t)). \tag{4}$$

Appendix B
The Signed Distance between two Spheres

Let there be two spheres, sphere A and sphere B. Let $\boldsymbol{a}(t) \in \mathbb{R}^3$ be the origin of sphere A and let $\boldsymbol{b}(t) \in \mathbb{R}^3$ be the origin of sphere B. The radii of the two spheres are denoted by r_A and r_B.

When the distance between the origins is equal to the sum of the radii, then surfaces of the spheres are touching each-other. To be explicit, let $\boldsymbol{\gamma}(t)$ be $\boldsymbol{b}(t) - \boldsymbol{a}(t)$. If $\|\boldsymbol{\gamma}(t)\|_2 = r_A + r_B$ then the surfaces of the spheres are touching.

Because $\|\boldsymbol{\gamma}(t)\|_2$ is defined as $\sqrt{\boldsymbol{\gamma}(t) \cdot \boldsymbol{\gamma}(t)}$, the derivatives are somewhat harder to derive. However, by squaring the formula we do get something that is usable:

$$\|\boldsymbol{\gamma}(t)\|_2^2 = (r_A + r_B)^2.$$
$$\|\boldsymbol{\gamma}(t)\|_2^2 - (r_A + r_B)^2 = 0.$$

Let us define the expression above as $s(t)$. When $s(t) = 0$, the surfaces of the spheres are touching. We only need to find the first occurrence of t such that $s(t) = 0$:

$$s(t) = \boldsymbol{\gamma}(t) \cdot \boldsymbol{\gamma}(t) - (r_A + r_B)^2. \tag{5}$$

Again we derive:

$$\dot{s}(t) = 2\dot{\boldsymbol{\gamma}}(t) \cdot \boldsymbol{\gamma}(t). \tag{6}$$

and

$$\ddot{s}(t) = 2\left(\ddot{\boldsymbol{\gamma}}(t) \cdot \boldsymbol{\gamma}(t) + \dot{\boldsymbol{\gamma}}(t) \cdot \dot{\boldsymbol{\gamma}}(t)\right). \tag{7}$$

Appendix C
Two Lines (in 3D)

Let there be 4 points in world-space: $\boldsymbol{a}(t)$, $\boldsymbol{b}(t)$, $\boldsymbol{c}(t)$ and $\boldsymbol{d}(t)$, and let $(\boldsymbol{a}(t), \boldsymbol{b}(t))$ and $(\boldsymbol{c}(t), \boldsymbol{d}(t))$ define ℓ_1 and ℓ_2 respectively. Let $\boldsymbol{\delta_1}(t)$ be $\boldsymbol{b}(t) - \boldsymbol{a}(t)$ and let $\boldsymbol{\delta_2}(t)$ be $\boldsymbol{d}(t) - \boldsymbol{c}(t)$. Let $\boldsymbol{n}(t) = \boldsymbol{\delta_1}(t) \times \boldsymbol{\delta_2}(t)$, be the direction in which the distance between the lines segments should be measured. The function $f(t)$ in Section IV-C is defined below as $s(t)$.

$$s(t) = \boldsymbol{n}(t) \cdot \gamma(t). \tag{8}$$

Where $\gamma(t) = \boldsymbol{c}(t) - \boldsymbol{a}(t)$. We can apply $\frac{d}{dt}$ on $s(t)$:

$$\dot{s}(t) = \dot{\boldsymbol{n}}(t) \cdot \gamma(t) + \boldsymbol{n}(t) \cdot \dot{\gamma}(t). \tag{9}$$

Let us now derive $\dot{\boldsymbol{n}}(t)$:

$\dot{\boldsymbol{n}}(t) = \frac{d}{dt}(\boldsymbol{\delta_1}(t) \times \boldsymbol{\delta_2}(t)).$

$$\dot{\boldsymbol{n}}(t) = \dot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) \times \boldsymbol{\delta_2}(t) + \boldsymbol{\delta_1}(t) \times \dot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t). \tag{10}$$

It is of course also required to state $\dot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t)$, $\dot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t)$ and $\dot{\gamma}(t)$:

$$\begin{aligned} \dot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) &= \dot{\boldsymbol{b}}(t) - \dot{\boldsymbol{a}}(t). && (11)\\ \dot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t) &= \dot{\boldsymbol{d}}(t) - \dot{\boldsymbol{c}}(t). && (12)\\ \dot{\gamma}(t) &= \dot{\boldsymbol{c}}(t) - \dot{\boldsymbol{a}}(t). && (13) \end{aligned}$$

Applying $\frac{d}{dt}$ on equation 9 yields:

$\ddot{s}(t) = \frac{d}{dt}\left(\dot{\boldsymbol{n}}(t) \cdot \gamma(t)\right) + \frac{d}{dt}\left(\boldsymbol{n}(t) \cdot \dot{\gamma}(t)\right).$

$\ddot{s}(t) = \ddot{\boldsymbol{n}}(t) \cdot \gamma(t) + \dot{\boldsymbol{n}}(t) \cdot \dot{\gamma}(t) + \dot{\boldsymbol{n}}(t) \cdot \dot{\gamma}(t) + \boldsymbol{n}(t) \cdot \ddot{\gamma}(t).$

$$\ddot{s}(t) = \ddot{\boldsymbol{n}}(t) \cdot \gamma(t) + 2\dot{\boldsymbol{n}}(t) \cdot \dot{\gamma}(t) + \boldsymbol{n}(t) \cdot \ddot{\gamma}(t). \tag{14}$$

Where $\ddot{\boldsymbol{n}}(t)$ is derived as follows:

$\ddot{\boldsymbol{n}}(t) = \frac{d}{dt}\left(\dot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) \times \boldsymbol{\delta_2}(t) + \boldsymbol{\delta_1}(t) \times \dot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t)\right).$

$\ddot{\boldsymbol{n}}(t) = \frac{d}{dt}\left(\dot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) \times \boldsymbol{\delta_2}(t)\right) + \frac{d}{dt}\left(\boldsymbol{\delta_1}(t) \times \dot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t)\right).$

Resulting in:

$$\ddot{\boldsymbol{n}}(t) = \ddot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) \times \boldsymbol{\delta_2}(t) + 2\dot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) \times \dot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t) + \boldsymbol{\delta_1}(t) \times \ddot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t). \tag{15}$$

For completeness, it is also required to mention $\ddot{\gamma}(t)$, $\ddot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t)$ and $\ddot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t)$:

$$\begin{aligned} \ddot{\gamma}(t) &= \ddot{\boldsymbol{c}}(t) - \ddot{\boldsymbol{a}}(t). && (16)\\ \ddot{\boldsymbol{\delta}}_{\boldsymbol{1}}(t) &= \ddot{\boldsymbol{b}}(t) - \ddot{\boldsymbol{a}}(t). && (17)\\ \ddot{\boldsymbol{\delta}}_{\boldsymbol{2}}(t) &= \ddot{\boldsymbol{d}}(t) - \ddot{\boldsymbol{c}}(t). && (18) \end{aligned}$$

Appendix D
Point-Plane Distance

Suppose that we are interested in the distance between a plane and a point. The distance between a plane and a point can be found by sampling three points, one in A-space, and two in B-space.

Let point $\boldsymbol{a}$ be our point in A-space. Let $\boldsymbol{b}$ be a point on the plane in B-space, and let $\boldsymbol{c}$ be a point in front of the plane, i.e. $\boldsymbol{c} = \boldsymbol{b} + \hat{\boldsymbol{n}}$, where $\hat{\boldsymbol{n}}$ is the normal of the plane. Now that we have defined $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ in their local spaces, let us consider these same points, but now they are transformed to world space. Note that now, these three points are time dependent.

We can now easily compute the distance between point $\boldsymbol{a}(t)$ and the plane in world-space:

$$s(t) = \boldsymbol{\gamma}(t) \cdot \hat{\boldsymbol{n}}(t). \tag{19}$$

Where $\boldsymbol{\gamma}(t) = \boldsymbol{a}(t) - \boldsymbol{b}(t)$ and $\hat{\boldsymbol{n}}(t) = \boldsymbol{c}(t) - \boldsymbol{b}(t)$. Applying $\frac{d}{dt}$ on equation 19, yields:

$\dot{s}(t) = \frac{d}{dt}(\boldsymbol{\gamma}(t) \cdot \hat{\boldsymbol{n}}(t)).$

$$\dot{s}(t) = \dot{\boldsymbol{\gamma}}(t) \cdot \hat{\boldsymbol{n}}(t) + \boldsymbol{\gamma}(t) \cdot \dot{\hat{\boldsymbol{n}}}(t). \tag{20}$$

Were $\dot{\boldsymbol{\gamma}}(t) = \dot{\boldsymbol{a}}(t) - \dot{\boldsymbol{b}}(t)$ and $\dot{\hat{\boldsymbol{n}}}(t) = \dot{\boldsymbol{c}}(t) - \dot{\boldsymbol{b}}(t)$.

Now, we take the derivative once more:

$\ddot{s}(t) = \frac{d}{dt}\left(\dot{\boldsymbol{\gamma}}(t) \cdot \hat{\boldsymbol{n}}(t) + \boldsymbol{\gamma}(t) \cdot \dot{\hat{\boldsymbol{n}}}(t)\right).$

$\ddot{s}(t) = \frac{d}{dt}\left(\dot{\boldsymbol{\gamma}}(t) \cdot \hat{\boldsymbol{n}}(t)\right) + \frac{d}{dt}\left(\boldsymbol{\gamma}(t) \cdot \dot{\hat{\boldsymbol{n}}}(t)\right).$

Resulting in:

$$\ddot{s}(t) = \ddot{\boldsymbol{\gamma}}(t) \cdot \hat{\boldsymbol{n}}(t) + 2\dot{\boldsymbol{\gamma}}(t) \cdot \dot{\hat{\boldsymbol{n}}}(t) + \boldsymbol{\gamma}(t) \cdot \ddot{\hat{\boldsymbol{n}}}(t). \tag{21}$$

Were $\ddot{\boldsymbol{\gamma}}(t) = \ddot{\boldsymbol{a}}(t) - \ddot{\boldsymbol{b}}(t)$ and $\ddot{\hat{\boldsymbol{n}}}(t) = \ddot{\boldsymbol{c}}(t) - \ddot{\boldsymbol{b}}(t)$.

Appendix E
Elaborate Test Results

See Figures 8, 9, 10 and 11.

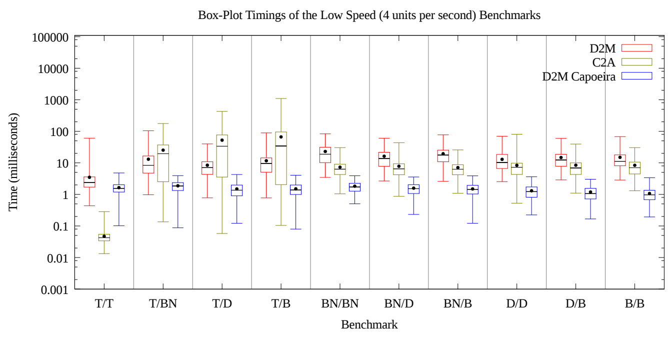

Fig. 8

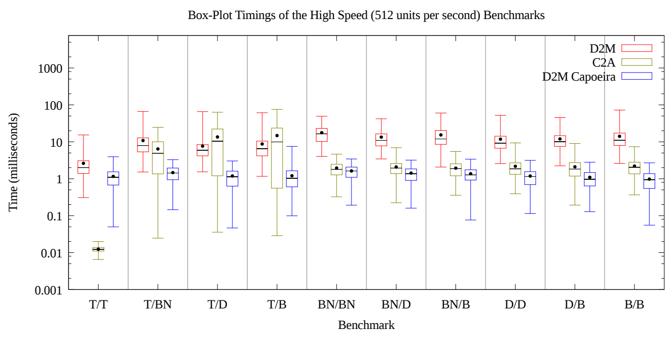

Fig. 9

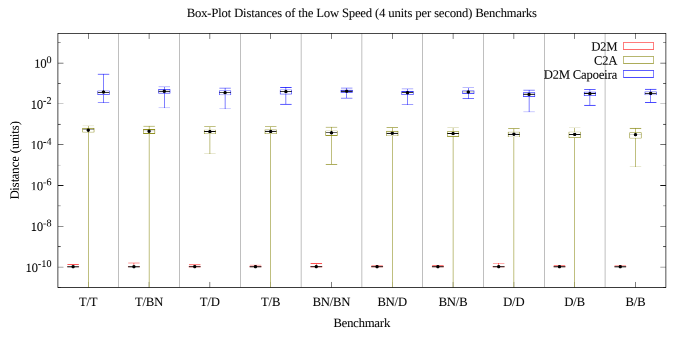

Fig. 10

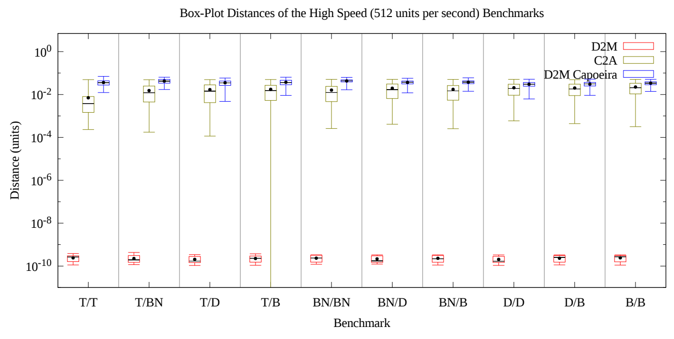

Fig. 11

References

[1] John Canny. Collision detection for moving polyhedra. 1984.

[2] Erin Catto. Iterative dynamics with temporal coherence. 2005.

[3] S. Redon A. Kheddar S. Coquillart. An algebraic solution to the problem of collision detection for rigid polyhedral objects. 2000.

[4] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

[5] David Eberly. Intersection of convex objects: The method of separating axes. 2008.

[6] Jens Eckstein and Elmar Schömer. Dynamic collision detection in virtual reality applications. In *University of West Bohemia*, pages 71–78, 1999.

[7] S. Gottschalk, M. C. Lin, D. Manocha, S. Gottschalk, M. C. Lint, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. ACM SIGGRAPH, 171–180*, 1996.

[8] Herman Haverkort. *Results on Geometric Networks and Data Structures*. PhD thesis, 2004.

[9] Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. In *International Journal of Pure and Applied Mathematics*, 2003.

[10] Brian Vincent Mirtich. Impulse-based dynamic simulation of rigid body systems, 1996.

[11] Ramanathan Muthuganapathy, Gershon Elber, Gill Barequet, and Myung-Soo Kim. Computing the minimum enclosing sphere of free-form hypersurfaces in arbitrary dimensions. *Computer-Aided Design Volume 43 Issue 3, March, 2011*, Computer-Aided Design:247–257, 2011.

[12] Arnold Neumaier. Taylor forms - use and limits. *Reliable Computing*, 2003:9–43, 2002.

[13] Stephane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Fast continuous collision detection between rigid bodies. In *Proc. of Eurographics (Computer Graphics Forum)*, page 2002, 2002.

[14] Stephane Redon, Young J. Kim, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection for articulated models. In *ACM Symposiumon Solid Modeling and Applications*, 2004.

[15] John M. Snyder. Interval analysis for computer graphics. In *Computer Graphics*, pages 121–130, 1992.

[16] Min Tang, Young J. Kim, and Dinesh Manocha. C2a: Controlled conservative advancement for continuous collision detection of polygonal models. *Proceedings of International Conference on Robotics and Automation*, 2009.

[17] Daniel J. Tracy, Samuel R. Buss, and Bryan M. Woods. Efficient large-scale sweep and prune methods with aabb insertion and removal. In *VR*, pages 191–198. IEEE, 2009.

[18] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. 1998.

[19] Xinyu Zhang, Stephane Redon, Minkyoung Lee, and Young J. Kim. Continuous collision detection for articulated models using taylor models and temporal culling, 2007.