Accurate and efficient continuous collision detection

Buddingh', W.L.B.

Award date:
2014

Accurate and Efficient Continuous Collision Detection

Wouter L. B. Buddingh'

January 2014

Words of Thanks

While writing this thesis, I received a lot of help from my supervisor Andrei Jalba. I would like to explicitly thank him for that. Next to this, I have received a lot of advice from people of the Visualization/Algorithms group of the TU/e. Although they are not mentioned explicitly, they have contributed indirectly to this work. Also, I would like to thank family and friends for the moral support they have given me.

Contents

1 Introduction

In the last decades, interactive simulation has gained popularity in many fields. Whether it is interactive simulation with respect to gaming or interactive simulation with respect to space simulation, an interactive simulation system will most likely require some form of collision detection. Another field which also benefits from collision detection is robotics. A collision detection system can be used to assure that a planned motion of one or multiple robot arms is collision free.

Collision detection is the process of determining whether objects intersect or will intersect within a given time-span. In real-life, collision detection (and collision response) is something we get for free (sometimes unwanted). That two objects cannot occupy the same space at the same time is obvious in real-life, but kind of exotic within a mathematical construct or computer program. Usually objects and especially their poses are determined by a collection of numbers. There is no straightforward way to make these numbers represent objects that do not intersect and behave in the same way as real objects do. The latter properties have to be enforced by a system.

Collision detection systems come in many flavours. It is possible to subdivide them into two categories:

- Static Collision Detection
- Continuous Collision Detection (or C.C.D.)

Static collision detection is the processes of determining whether two objects intersect at a given moment in time. This question can be answered with a simple yes or no.

With continuous collision detection, one has to specify the behaviour of objects over a given time-span. A system that preforms C.C.D. does not only compute whether objects will intersect within the given time-span, when objects do intersect, it also computes the first time of intersection. This moment in time is commonly referred to as Time of Contact or TOC. Whenever the TOC is known, this also determines additional traits of the collision, like the configuration of the bodies at impact. This is because the configuration of the objects is defined as a function of time.

Why do we need continuous collision detection? Sometimes static collision detection is not good enough. Conceptually, a C.C.D. system is able to guarantee that no collisions are missed, whereas this is not the case with a static collision detection system. Static collision detection systems suffer from an effect called "tunnelling" (see Figure 1). Which means that it behaves in a counter intuitive way. E.g. an object may pass trough a solid wall, because the *instance of collision* is missed due to a low number of samples (static collision tests) per time unit. This problem can be partially solved by increasing the number of samples per time unit (adaptively), in such a way that the tunnelling effect is almost gone. In order to solve this problem completely, C.C.D. is required.

Objects can be three-dimensional or they can be two-dimensional. A 2D-object is usually an abstraction of a 3D-object. Certain problems are better modelled in 2D because in some cases the third dimension adds little or nothing. However, the 3D case is analogous to the real-life situation. In real life, objects can move, and therefore they can collide. Objects can deform, break

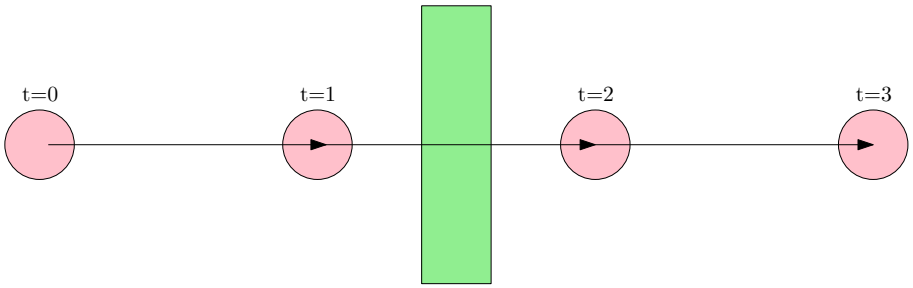

Figure 1: A collision that is missed.

and melt. The cases mentioned in the previous sentence and other exotic transformations on objects are explicitly not treated within this thesis. *Rotation* is also considered to be some sort of movement. Certain collision-detection systems are constricted to movement without rotation. Rotations usually make the process of collision detection somewhat harder, and it is unnatural to leave it out. Because of the latter, rotations are an integral part of this thesis. As can be concluded from the text above, objects are not allowed to deform. In many cases, this sort of behaviour may be regarded as unnatural. However, there exists quite a large class of problems that can be modelled by only considering non-deformable objects. These type of objects are commonly referred to as rigid-bodies.

Rigid bodies are also considered to be impermeable. When two rigid (impermeable) objects approach each-other with some relative velocity between them, the collision needs to be resolved with an impulse response. Applying forces over a period of time would cause inter-penetration of objects. By using impulse, the objects will instantaneously change their velocities, causing the objects not to penetrate.

There are still some other properties of the objects that are relevant, namely their type of motion, and the class of their shapes. The shape of an object can be of a certain type. For example a spherical object is of the type "sphere", whereas a cube is of the type "convex polyhedron". It is possible to define shapes in a variety of ways. When it comes to curved shapes, there is no simple universal way to define them all. This thesis will consider *polygon soups*. Polygon soups can define any bounded non-curved shape. Thus, a broad range of geometrical objects are considered within this thesis. Polygon soups are always defined in three dimensions. The two-dimensional equivalent of a polygon soup is edge-set (informally edge soup).

The motion of an object also needs to be restricted. To be complete it should be stated that time and space are organized according to the classical Newtonian view of the world. The term "world" is often used to describe the virtual world in which a simulation resides. When considering 3D collision detection, the world is represented by four dimensions. When considering 2D collision detection, the world is represented by three dimensions. It should be noted that the 3D problem is not fully four-dimensional (in the Euclidean sense), since only moments in time, or time-spans are considered and not arbitrary 4D-space intersections. This drastically simplifies things. In Figure 2 a 3D space (x,y and time) is projected onto the plane.

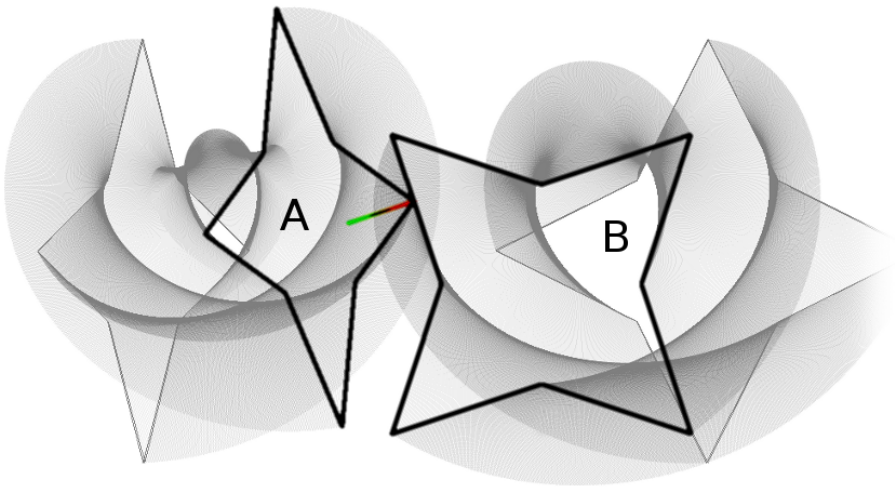

Figure 2: A 2D collision that was detected.

Within the context of this master project, a 3D, highly accurate, continuous collision detection system has been devised. When considering the units of the system to be meters, and when considering the object size to be in the order of one meter, the system has nanometer precision. This means that when the objects are declared to be in contact, there is a space in between them of less than a nanometer. Though the system is very precise, when considering complicated models (models having a complexity of 70K-100K triangles), it has a performance that is (worst-case) comparable to a reference system; Controlled Conservative Advancement [20] when considering independent collisions. The efficiency of the system that was devised is due through the usage of efficient geometric indexing.

It can be said that for many purposes continuous collision detection is not required to be accurate. For a certain class of applications, very little precision is required. As a by-product of constructing the highly accurate method, a much less accurate method has been found. We shall call this method *Capoeira mode*. The method is named after a Latin-American martial art in which non-contact is important.

Capoeria mode is less accurate than the reference system, though considerably faster when collisions are independent. The notion of a non-independent collision is reasonably artificial. Usually a collision stands by itself, but suppose that two consecutive collisions are almost identical, the collision detection process can be accelerated. However, this notion is not exploited by the systems devised within this thesis.

The goal of this thesis is to illustrate how to compute the instance of collision, given two rigid bodies under some continuous motion. The problem of computing the instance of collision between more than two rigid bodies can be reduced to the problem of computing the instance of collision between two rigid

bodies.

Within this thesis, a certain motion type is used namely *linear translational and linear rotational motion*. The motion type is reasonably elementary, yet from a mathematical point of view it is already difficult to solve. A motivation is given why a customized numerical method is used in order to solve the instance of collision (of a feature pair). Because a numerical method is used, this gives the system more flexibility. *In theory*, the motion type of the system is customizable, under the pre-condition that the customized motion type is sufficiently smooth ($\mathcal{C}_2$ continuity of all point trajectories is enough). Some other preconditions may apply here too.

In this thesis, the following contributions are made:

- Global level contributions
  - A highly accurate C.C.D. system is presented.
  - A very fast system that is less accurate (Capoeira mode) is described.
- Lower level contributions
  - A root finder is introduced which is capable of finding the first root of any $\mathcal{C}_2$ continuous function.
  - A custom Bounding Volume Hierarchy is introduced that is effective.

This thesis is organized as follows: In Chapter 2, the notations are partially explained. Next, some background information is given in Chapter 3. Afterwards, in the related work section, some of the accomplishments of others are described (Chapter 4). A method based on feature testing is described in Chapter 5. This method can be applied in both 2D and 3D. The 3D system is benchmarked in the result section. Chapter 8 indicates potential research topics and discusses them in relation to this thesis. In Chapter 9, this thesis is concluded.

2 Definitions

This chapter introduces the notation that is used, throughout the thesis. Big-Oh notation is denoted by $\mathcal{O}(x)$. It can either be infinite asymptotic or infinitesimal asymptotic. Whenever it is used, the context will indicate the intention.

The symbol $\mathcal{C}_n$ is used to denote the continuity class of functions. With $\mathcal{P}(S)$ the power set of set S is denoted.

The $\cdot$ symbol is used in different contexts. It usually describes the dot product but it can also describe normal multiplication of objects. The $\times$ symbol specifies the 3-vector cross product, but sometimes it is also used to denote the number of entries of a matrix e.g. $\mathbf{A} \in \mathbb{R}^{2\times 2}$, sometimes it denotes the Cartesian product.

Time is always a real value and described by the letter t. The span of a time-interval is denoted by δt. The set of intervals is denoted with $\mathbb{IR}$ and an element of this set is written with a lower case, non bold letter. An interval is a pair of two real numbers $[a, b]$ for which $a \leq b$.

The term TOC denotes Time Of Contact, and usually describes the first Time of Contact. However, the term is context sensitive; sometimes multiple (distinct) TOCs are used within the same context and thus not all of them can be the first.

It is often the case that a second order polynomial is designated with the name parabola. Whenever the name parabola is used, this may just as well designate a hyperbola, or it might even designate a line.

The set of quaternions is denoted by $\mathbb{H}$. Quaternions [9] are hyper-complex numbers and they are used in order to define rotations of objects. Additionally scalars are written with a lower-case, non-bold letter e.g. s. Vector quantities are written using a bold lower-case letter e.g. $\mathbf{v}$. Matrices are denoted by using a bold, upper-case letter and quaternions are denoted by a bold, lower-case letter. The variable names i, j and k are used for the imaginary identities of the quaternions. Next to describing the imaginary identities of the quaternions, the letters i, j and k can also be used as indices. Selecting an identity of a quaternion is done with a subscript function e.g. $\mathbf{q}_i$ or $\mathbf{q}_r$. The latter example ($\mathbf{q}_r$) selects the real component of $\mathbf{q}$.

3 Background

This section describes aspects like motion and some general approaches to solving the collision detection problem.

3.1 Motion

Objects that collide do so because they are moving. A rigid object follows a certain trajectory. To be more precise, each point of a rigid object follows a certain trajectory. Rigid objects can translate, i.e. change their position while time passes, and they can rotate. From now on, the term rigid will usually be left out, because this thesis does not consider non-rigid objects.

If objects are modelled in such a way that they are only able to translate, the collision detection process is drastically simplified. As said in the introduction, in this thesis a more generic case is considered, including both translational and rotational motion.

The next step is to describe various kinds of motion that are both translating and rotating. Let us constrict ourselves to *analytical functions*. The reason for this is that it is possible to use Taylor's theorem in order to reason about these functions. Analytical functions can be differentiated infinitely many times around any point. Such functions are called smooth functions. In contrast, smooth functions are not always analytical functions [1]. Each point of an object has a trajectory in space-time. In 3D, this path can be described by using three one-dimensional Taylor series. Such a construct is usually called a parametrized function. One parameter (i.e. time) is supplied to the function and a three-dimensional positional vector is returned.

Because the motion of the object contains rotations, the Taylor series is therefore based on sine and cosine expressions. Note that the Taylor series representation of the sine/cosine contains an infinite number of non-zero terms.

The Taylor series can be truncated in order to get an n^{th}-order Taylor polynomial that approximates the curve.The approximation quality is described by using (infinitesimal) asymptotic (big-Oh) notation (see Chapter 2).

Note that we have been describing the trajectory of any point in an object. Each object has a special point namely its centroid or center of gravity. A special feature of this point is that its trajectory is not affected by the rotation of the object. In fact in some cases, the trajectory of this object can be described exactly by a polynomial. Assuming that the object is undergoing some complicated motion e.g. its motion is governed by some differential equation, then the motion is most likely described by a Taylor series having an infinite number of non-zero terms. Anyway, in order to be able to type the motion, it is handy to truncate this Taylor series.

Next to the center of gravity, the rotation of an object is usually described by some variable. In 2D this variable can be a scalar representing an angular value. In 3D this variable can be a hyper-complex number called a quaternion [9]. Regardless of the representation of this variable, this variable can change. Therefore, it also has a rate of change. And a rate of change of the rate of change (and so on). Similar to the trajectory of the centroid, the value of the

[1] `http://en.wikipedia.org/wiki/Non-analytic_smooth_function`

rotational variable can be described by a Taylor series. Again we might wish to truncate this Taylor series.

It should be noted that the behaviour of rotations in 3D is a bit different than the behaviour of rotations in 2D. For instance, the quaternion Taylor series of a linear rotation has infinitely many non-zero terms, while in 2D, the Taylor series for a linear rotation is just a linear function (depending on the representation).

3.1.1 Linear Translational and Linear Rotational Motion

We can make a first order simplification of the general motion of an object. This first order simplification is not a truncation of the Taylor series of each point of the object. If one would do so, then each point of the object would follow some linear motion, causing the object to deform in an unnatural way (if it were to rotate). In Section 3.2, the concept of continuous motion is explained. For now let us briefly state that within the context of this thesis, continuous motion implies rigidity and all of the points of the object to have $\mathcal{C}_0$ continuous trajectories.

If we basically truncate all Taylor series of all points of the object, we get linear motion in the case of non-rotation, and something funny when the object is rotating. Rotations are basically two-dimensional operations. In 2D, an orientation is basically a scalar value called α. A rotation, $\delta\alpha$ can be used to increment the rotation scalar α. The alpha value can be converted into a 2D rotation matrix that is capable of transforming points that are non-rotated into points that are rotated. In 3D, an orientation cannot be represented by a single scalar. An orientation in 3D can be represented by a 3×3 rotation matrix. By rotating each column vector of the orientation matrix, one is able to change the orientation. A rotation is defined around a certain (normalized) axis. When the axis is pointing towards the observer, one can define the amount of rotation ($\delta\alpha$) as being counter clockwise around the axis. Usually the rotation is measured in radians.

Rotating the column vectors of an orientation matrix in 3D around a certain axis is a non-trivial operation. In [24] the authors give a formula that describes the motion that is discussed here. This formula uses 3×3 matrices in order to describe the transformation. In Appendix A the same transformation is described, but instead of matrices, quaternions are used. Both definitions are equivalent to the informal definition given below:

To summarize the concept of linear translational and linear rotational motion; the center of gravity of an object under such motion follows linear motion. While the body gets rotated around the center of gravity along an axis with a fixed orientation having constant rotational velocity.

It must be noted that this type of motion does not necessarily represent the motion of an object under the absence of net external force/torque. Newton's first law states that the velocity of an object does not change when no net external force/torque is applied. This implies that linear momentum is conserved. Likewise, angular momentum also happens to be conserved in the absence of net external torque. However, conservation of angular momentum, does not imply a constant rotational velocity around a fixed axis. In reality, the axis of rotation of an object may change due to time-varying distribution of the mass.

In Figure 3, the velocity of a point trajectory at t_0 is shown. The user of the system is free to choose the variables ω and $\mathbf{v}$. If these variables are chosen such

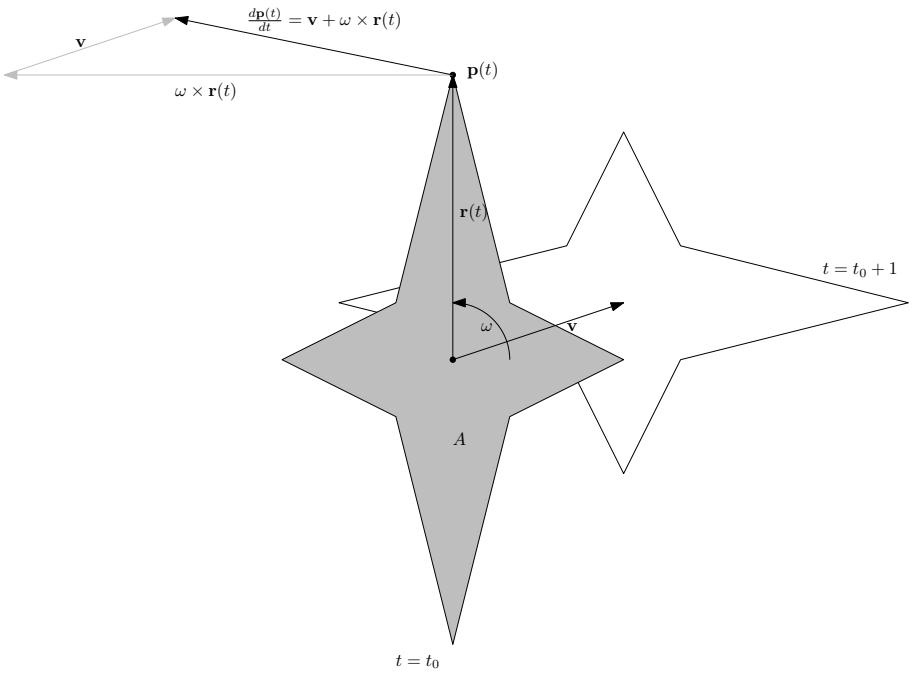

Figure 3: A sketch illustrating the velocity of a point under linear translational and linear rotational motion.

that they match the real angular velocity and translational velocity of object A at t_0 then the velocity of each point $\mathbf{p}(t_0) \in A(t_0)$, matches the velocity of the associated real trajectory. Because all the trajectories have an error of $\mathcal{O}(\delta t^2)$ (when considering their Taylor series) we can state that all trajectories are a first order approximations of the associated real trajectories.

3.1.2 Linear-Interpolation Motion

Given a beginning pose i.e. an orientation and a location, and an ending pose. The motion could be defined by preforming linear interpolation on the center of mass and by preforming some form of linear interpolation on the orientations. For the interpolation of orientations one would pick quaternion spherical-linear-interpolation (or SLERP), because it is the most natural type of interpolation. Note that this motion is very similar to linear translational and linear rotational motion. Because the amount of rotations per motion segment is limited with linear-interpolation motion, it is not equivalent to the motion type expressed in the previous section. Linear-interpolation motion is the type of motion that is sometimes used when dealing with conservative advancement [20] (a technique described in Section 3.3.2). A time domain in which a single linear translational and linear rotational motion exists can be translated into multiple linear-interpolated motion segments (featuring quaternion-SLERP). Why the two motion types are not equivalent is just a matter of differing definitions. However, they can be converted in both directions.

3.1.3 Ballistic Motion

In [15], Mirtich uses quadratic polynomials in order to describe the trajectory of the centroid of rigid bodies.

In a rigid body simulation, forces are sampled at a certain moment in time, and they are dependent on the state of the simulation (which includes aspects like velocity and angular velocity). Using a second order Taylor approximation of the trajectory of the centroid, i.e. an approximation based on position, velocity and acceleration seems to be a logical choice.

However, this type of approximation does not seem to be used much. Instead of using this type of approximation, an Euler approximation is used more frequently (since it also converges to the exact solution when the time-step converges to 0). Alternatively some higher order scheme is used, e.g. some instance of the Runge Kutta family of integrators.

3.1.4 Articulated Motion

Earlier in the introduction, the example of a robot arm was given. A robot arm moves around a joint. This is usually a rotating joint or some translating mechanism. The translations are not the problem, since translations are directly related to the Cartesian system; the rotations cause difficulties. In the ideal case, exact articulated continuous collision detection can be preformed. However, as the number of joints in the robot arm grows, the more complex the motion of the robot arm becomes. Increased complexity means that more processing power is required. When considering a robot arm, the "approximation" can be very precise. The trajectory of the robot arm is precisely defined (up to arithmetical precision). It is possible to compute the precise time of contact of the robot arm with e.g. another robot arm, up to a specified time epsilon (ignoring arithmetical errors). If the arithmetical precision of the system allows for it, it is even possible to approximate up to a distance epsilon (the application of epsilon values will be discussed in Section 5.3).

To summarize the above, the model is analytical, but the solution has to be approximated using numerics. In theory it is possible to give an arbitrary accurate approximation. The same holds for other types of motion if they are considered to be exact instead of approximate.

However, in Chapter 4 others are aiming at something quite different than giving an arbitrary accurate approximation. Their main target is speed, and not approximation accuracy. The reasoning behind this is that a time step, i.e. a single motion segment, is only persistent over a small period of time. This is the reason why many care less about the asymptotic accuracy of motion within a timestep. The current subject is closely related to the following section.

3.2 Truly Continuous Motion

When making a simulation, one naturally starts by animating a static scene. Usually, the simulation is defined in frames. A frame usually lasts for a time interval, lets say $[0, \delta t)$. However, frames are often treated as if they happen at a single moment in time. It would be strange to say that this approach is wrong, and actually it is not. We need to abstract in order to make things work. But if there is a frame at $t = 0$ and a frame at $t = \delta t$, then an infinite amount of

frames are located at $(0, \delta t)$. It can be stated that in some cases the abstraction is wrong.

Collision detection systems that operate at discrete frames, are called discrete-time collision detection systems. C.C.D. systems have to obey some properties in order to be called "truly continuous" [18].

These properties can be described in an intuitive way:

1. Objects are not allowed to "teleport", or "jump" in time
2. Objects are not allowed to deform

These properties can also be described formally. The first property is defined as follows:

Let $A(t) \subseteq \mathbb{R}^3$, be a time-dependent point-set representing an object. (To be precise: $A : \mathbb{R} \mapsto \mathcal{P}(\mathbb{R}^3)$.) Let $\mathbf{p}(t) \in A(t)$ and let $\mathbf{p}(t+\delta t) \in A(t+\delta t)$ where $\mathbf{p}(t)$ is a parametrized function describing the trajectory of a point in object A. Then the Euclidean distance between the two points converges to zero as δt goes to 0. Summarized in a compact way:

$$\lim_{\delta t \downarrow 0} \mathbf{p}(t) = \mathbf{p}(t + \delta t) \tag{1}$$

The definition above does not suffice yet, since deformable objects e.g. (incompressible) fluids also obey the above. Property 2, can also be formalized:

> There exists a vector $\mathbf{o} \in \mathbb{R}^3$ and a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ such that for each point $\mathbf{p} \in A(t)$ the point $\mathbf{R} \cdot \mathbf{p} + \mathbf{o} \in A(t + \delta t)$ and for each point $\mathbf{q} \in A(t + \delta t)$ the point $\mathbf{R}^{-1} \cdot (\mathbf{q} - \mathbf{o}) \in A(t)$.

Note that the second transformation, the transformation of point $\mathbf{q}$, is the inverse transformation of the one applied on $\mathbf{p}$. The second part is there because the relation is bijective.

Both properties symbolize part of the motion of a real object. At least the motion that most people consider real, considering the classical Newtonian view of the world.

To give an example, an object that has truly continuous motion can never pass trough a (rigid) hole if it does not fit trough the rigid hole. This might seem obvious, but with discrete collision detection, this may not be the case.

Some discrete systems are likely to miss collisions. However the property above is not only about not missing collisions, it is about avoiding physical impossibilities. It should be noted however, that the truly continuous property does in no way account for all physical impossibilities. There is nothing that guards for jump discontinuities in the first derivative. In reality this is impossible but actually, this is what is preferable. As explained in the introduction, the objects are rigid and the rigid collisions must be resolved with impulse. So the first derivative of the object can have irregularities.

Various authors of literature on (truly) continuous collision detection make up different kinds of motion that have little to do with being realistic, but can still be labelled as "continuous". These types of motion are constructed in such a way that they are easy to compute. Section 4 will elaborate on this.

Let us discuss one example of a strange continuous motion. Suppose that object A has to go from pose a to pose b. A way to accomplish this is to

first rotate the object in such a way that it will assume the orientation of b. This rotation can happen in time interval $[0, \frac{\delta t}{2})$. Afterwards the object can be translated to its final position (within time interval $[\frac{\delta t}{2}, \delta t)$).

Now suppose that there is a collision between object A and some other object in interval $[0, \delta t)$. The time of the first moment of contact retuned by the collision detection system, will be very inaccurate. Said differently, the approximation accuracy of motion within the interval is of $\mathcal{O}(1)$ (i.e. an error of $\mathcal{O}(\delta t)$, or stated differently 0^{th} order accurate). In contrast, (simultaneous) linear translational and linear rotational motion has an error of $\mathcal{O}(\delta t^2)$, due to its correspondence with explicit-Euler integration. These asymptotic error descriptions apply on every point of the object (see 3.1.1). For the case of the strange motion, in general, most likely all points will have a first time-derivative that is different from the first time-derivative of their actual trajectories. This conflicts with being $\mathcal{O}(\delta t)$ accurate.

3.3 Common Solutions

The most straightforward solution to continuous collision detection seems to be *feature testing* [6] [8] [11] [18]. This is the approach that will be used in this thesis. Other approaches include *conservative advancement* [15] [24] [20] [25], 4D-intersection-testing [5], swept-volume [13] and the GJK-raycast algorithm [23]. The latter approach does not consider rotations. Whether swept-volume is truly continuous depends on the implementation.

With 4D-intersection-testing, the problem is treated as if it were fully four-dimensional. In its basic form, 4D-intersection-testing only deals well with translations. Rotations have to be "hacked" into the system in a non-efficient way. Also such a hack will cause the system to strictly be non-continuous. Therefore in practice, 4D intersection only deals well with translational motion.

Conservative advancement is a reasonably new approach to continuous collision detection that is fast and effective. Originally, it was defined for convex shapes only, but recently work has been done on non-convex implementations. More on this will be described in Chapter 4.

Let us first start with the approach that will be used in this thesis, namely feature testing. However, the subject will be discussed briefly, because it will be made explicit in Section 5.1 and 5.2.

3.3.1 Feature Testing

Two objects, whether they are defined in 2D or in 3D, have features such as points and edges (and facets). Let a be a feature of object A and let b be a feature of object B. These features may collide or they may not collide. It is disputable whether a point and another point can collide. Although the probability of such an event happening is 0, when assuming a random initial configuration. When using finite precision arithmetic, this probability becomes a bit larger, however this is not the issue. Let us suppose that feature a is a point and feature b is a facet. A point and a facet can definitely collide. One has to devise a continuous test that is capable of determining the first time of contact of these features. Note that the word first is used, since the features may separate and collide again. It should again be noted that the term *time of contact* is abbreviated with TOC. Usually, this term also implies the first time

of contact. Before we give some indications on how to devise a continuous test, let us first complete the picture.

For each feature pair, one feature belonging to A and one feature belonging to B, compute the TOC. When a list of TOCs is computed, take the first one and this TOC will represent the collision time between object A and B. To be complete, not every feature pair has to collide, so when there is no collision, the TOC can either be infinite, or it can be excluded from the list. The story above describes part of the 3D case. The latter sentence says "part of" because edge/edge intersections were not explicitly mentioned. For a 2D implementation, a function that can compute the TOC between an edge and a point suffices.

Note that the story above says noting about motion. In principle the motion can therefore be generic (higher-order for example). Without considering the type of motion, the approach is already slow. The asymptotic time complexity is $\mathcal{O}(A_f \cdot B_f)$, where A_f and B_f represent the number of features of object A and B respectively. The latter statement assumes that the time complexity of a single feature test is constant. Let us now give some intuition on how to construct a feature test.

It is possible to construct a distance function between feature a and feature b. This distance function is a mapping from time to distance: $s : \mathbb{R} \mapsto \mathbb{R}$. Note that it is quite okay if distances are negative.

Lets consider the 2D case of a point and an edge for now. Basic computational geometry enables us to compute the distance between a point and a line segment. Before we do this, we first have to integrate the poses of both bodies to the right moment in time. The point and the line segment are defined in local coordinates of the associated object. When a moment in time is given, the coordinates of these features can thus be described in global (or world) coordinates by integrating the poses of both objects. Thus for each moment in time, we can compute the poses of the features and thus the distance in-between them. Now, we have our distance function. When the distance function reaches 0 (at a certain t) there is a collision. These collision points correspond to the roots of the distance function s.

The distance function above can be considered a bad example for two reasons:

- The function does not cross the time axis, it merely touches it.
- The distance derivative for a vertex on the edge is actually undefined (contradicting $\mathcal{C}_1$ continuity).

We can derive the first derivative analytically (at moments in time where the features are not intersecting). One might argue that it is possible to apply a root finding technique, such as Newton-Raphson iteration. However, usually the complexity of the motion disallows using this technique (disregarding the fact that the derivative is not well defined at every moment in time). Also the case distinction might spoil finding the roots. Likewise, a method similar to Newton-Raphson, called the Secant method will also not work, due to similar reasons.

The curvature of the function is just too complicated. The function may incorporate oscillations, oscillations having an arbitrary frequency (the oscillations are due to the rotations of the objects). In the case of linear translational

and linear rotational motion (see Figure 4, the red curve), it might be possible to chop the curve into parts (with respect to the time variable), that are of limited complexity, by looking at the oscillation with the highest frequency. However, this does not guarantee that a chopped part contains at most one root and that its curvature obeys the criteria of the selected "elementary" root finder. Note that the curvature cannot be chopped at the extremes because finding the extremes is even more difficult than finding the roots.

Using the root finder presented below, there is no need for the function to be $\mathcal{C}_1$ continuous.

Interval Arithmetic

There exists a simple and robust root finding method called *binary bisection* on intervals that is based on interval-arithmetic. This method is quite capable of finding the roots of any $\mathcal{C}_0$ continuous function, no matter its complexity. There are two downsides to this approach. The method has a moderate convergence speed, and it is inherently based on case distinctions, making it unable to fully exploit the huge processing power of modern-day computers. (Conditional jumps usually cause some lag in the execution of code.) More on the binary bisection method on intervals will be described in Section 5.1.

Taylor-Lagrange based Method

Part of this master project assignment is to find an alternative for using binary bisection on intervals. The alternative should be fast, instead of being logic based (like interval arithmetic), the alternative approach uses a combination of algorithm and calculus in order to find the roots. A similar approach can be found in [15], however it differs considerably form the method presented here.

In order for a Taylor based method that is presented within this thesis to work, the distance function needs to be $\mathcal{C}_2$ continuous. This immediately yields a problem, because our edge/vertex distance function does not obey this criterion. The problem can be solved by re-designing the entire feature test in a similar way as done in [11]. Instead of computing the distance between a vertex and an edge, it is preferable to compute the distance between a vertex and a line (this yields a signed distance). Such a computation can be done with ease, and when considering linear translational and linear rotational motion, we conjecture that all the derivatives of the distance function will be continuous.

The distance $s(t)$ is shown in Figure 4. The red curve represents the distance, while the grey blue curve represents the derivative of the distance. The derivative is illustrated in order to show that the derivatives of the function do not simplify. This property makes root-finding somewhat more difficult. Because the derivatives do not simplify, it is not easy to select an increasing or decreasing domain, that is why it is not possible to apply some variation of the Secant method.

Let us for now assume that we have a working root finder, see Section 5.2.1 for details. However, too many roots are found. One must also consider roots that correspond to the vertex hitting part of the line that lies outside of the domain of the edge. Before a root is reported, a static intersection query is performed to check whether the vertex is indeed inside the domain of the edge.

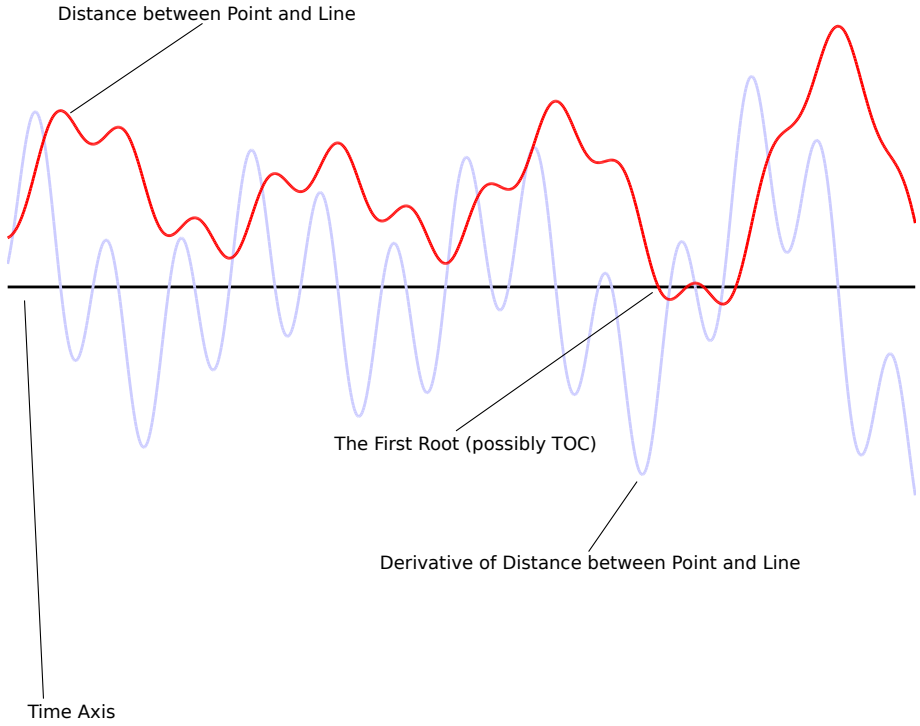

Figure 4: The signed distance ($s(t)$) between a point and a line both subject to linear translational and linear rotational motion.

If inside, the root is reported as a real intersection, if the vertex lies outside, the root is discarded.

Summarizing this method: brute force all feature combinations by using a continuous TOC test that is based on a distance function. Brute-forcing sounds bad, however multiple techniques exist to reduce the computational effort. The most prominent of those is by using bounding-volume-hierarchies or BVHs, see Section 3.4. BVHs is certainly not limited to feature testing. It describes a set of data-structures that preform spatial partitioning in a hierarchical way.

3.3.2 Conservative Advancement

Conservative Advancement is a technique that relies on static proximity queries in order to preform continuous collision detection. It was introduced by Brian Mirtich in [15]. The method operates on two convex objects.

In figure 5 the "maximal axis of separation" is shown. If one would rotate this axis, it would lose the property of being maximal. Such a maximal axis is defined for each pair of objects A and B. When the two objects are projected on the axis, the separation of the projections of both objects is maximal.

The distance the objects travel towards each other, can be measured along this axis. To make this a bit more specific, the distance object A travels on $\mathbf{a}$ (the axis), can be bounded by a linear equation, which is a mapping from δt to distance. The same holds for object B. By subtracting the two equations, we get a single equation. This equation states how far the objects have approached

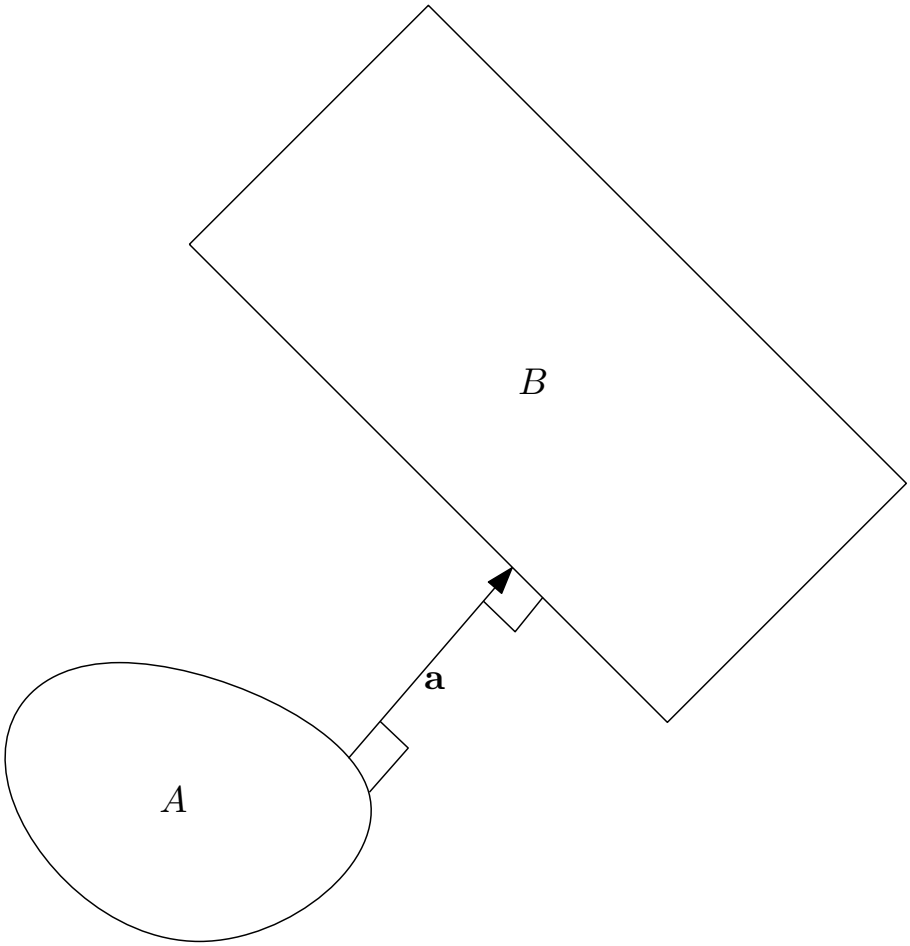

Figure 5: The line segment in-between the objects is the maximal axis of separation.

each other when given a certain δt. Note that the equation is a parametrized upper bound for how much the objects have approached each-other. We thus get a linear equation $f(\delta t) = a + b\delta t$. We then solve δt in $f(\delta t) = d$, where d is the distance between both objects (or the length of the axis). Once δt is determined, both objects can be safely integrated to $t + \delta t$, without the objects touching each other. At this moment in time, a new axis is computed, and the process starts again, until the distance between the objects drops below a certain threshold.

A property of this construct is that the objects will not overlap each-other. If the motion bounding function is constructed in the right manner, the method is continuous.

3.3.3 4D Intersection Testing

This method is well suited for linear translational movement of polyhedral objects. When rotations are required, a naive solution would be to use a swept-

volume-like strategy (see Section 3.3.4) in combination with this method. Better approximations of rotations seem to be possible, yet difficult.

For translational movement of polyhedral objects, the problem can be solved analytically. For the linear translational part, the method works by extruding a polyhedral representation into the time dimension. Note that this extrusion may be non-orthogonal, in fact the extrusion is non-orthogonal if the object is moving. When an object is extruded, an extra dimension is added, in this case this is the time dimension. Normally an object is a subset of $\mathbb{R}^3$. By extruding the object it becomes a subset of $\mathbb{R}^4$ (the set of all events). Let $A : \mathbb{IR} \mapsto \mathcal{P}(\mathbb{R}^4)$ and let B also be such a function. The 4D-volume like quantity of these sets is finite. However, the latter remark is of lesser importance, because there exists software that is quite capable of dealing with unbounded polytopes.

Let $t \in \mathbb{IR}$ be a time interval and let $A(t) \cap B(t)$ be the space-time intersection of these sets. If the intersection is empty, the objects do not collide within the interval t. If the intersection is non-empty, then we are interested in the set of events that are first.

Intersecting sets is nice, but how can this be done efficiently? Recall that the shape of the objects is polyhedral, and a polyhedral object can be decomposed in convex objects. Now that we have a set of convex objects (in $\mathbb{R}^3$), we also have the half-spaces that bound these objects (3D half-spaces are usually represented by using a 4-vector).

Each half-space can be extruded on its own. Extruding a half-space from 3D to 4D is a relatively simple operation. Such an extrusion requires one parameter, namely the velocity of the object. Now that we have a set of 4D-half-spaces for each convex polyhedron, we can construct a 4D-convex-polytope for each convex polyhedron. This convex-polytope is preferably also bounded by the 4D-half-spaces $t \geq 0$ and $t \leq \delta t$. The latter remark may help when one wishes to optimize the computations.

In Figure 6, a rectangle in the plane is given a linear motion up to a certain moment in time. After this moment in time, the motion is changed into another direction.

Since we have two objects we can brute force intersect all the (a, b) pairs of 4D-convex-polytopes. The intersection of 2 polytopes can be achieved by combining their half-spaces. This also holds in 4D (by definition). Thus for each 4D-convex-polytope pair (a, b) we get a resultant 4D-convex-polytope that represents part of the collision event set. Uniting all these event sets yields the total collision event set. We pick the most early events form this set. These can be considered to represent the collision surface at $t = TOC$.

Compositing a 4D-convex polytope is not easy. However, it is possible to construct a framework based on mathematical definitions that does just that. There is a more efficient way of solving the problem, that is by using (4D-)linear-programming (assuming that this can be done fast). For each (a, b) pair, a linear program is ran. And the objective function is to get an event as early as possible. The software that facilitates linear-programming should be able to indicate whether such an event exist (or perhaps even return a range of such events). The latter is of lesser importance, since one contact point is usually sufficient. Anyway, alternative contact points can be found differently (see Section 6.1).

With or without using linear programming, the process seems reasonably in-efficient. One might think of using 4D-bounding-boxes to bound the initial

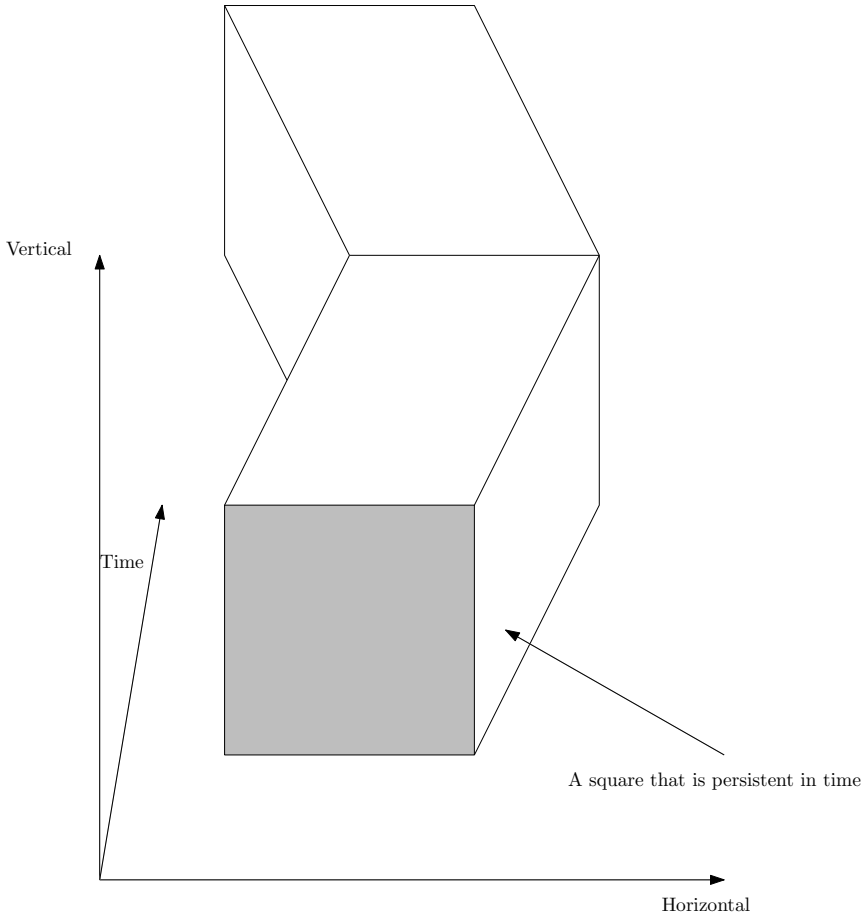

Figure 6: A square in the plane extruded into the time dimension.

4D-polytopes, and this bounding-box structure can be hierarchical. Another option is to use Binary-Space-Partition trees in order to cut down on the quadratic complexity. Similar approaches are summarized with Bounding Volume Hierarchies (see Section 3.4). One might have noticed the term "quadratic complexity". Why is the complexity of such an algorithm quadratic? This is because in the analysis, we assume that the complexity of a single convex polytope (in 3D or in 4D) is bounded by a constant. In fact, this is always the case if the polyhedral object is decomposed into tetrahedrons.

How to deal with rotations

This is the point where the method will become not strictly continuous any more. Non-continuous in the sense that the object is grown in order to guarantee that no collisions are missed. By growing the object (and thus deforming it), false-positives may be returned. It is possible to construct a system in such a way that false-negatives are impossible. An application of such a system can be robot-motion-planning.

The difficulty lies in finding a set of events that includes the events of the

rotating object. Preferably, this set is equal to the set of events that define the moving object while it is rotating, but this is not possible when using a finite number of 4D half-spaces.

Actually, the method is similar to swept-volume, which will be discussed in Section 3.3.4. Approximate rotations can be integrated in the 4D-framework though.

3.3.4 Approximate Swept Volume

Another technique that can be used is approximate swept volume collision detection. This method does not match the definition of truly continuous motion given in this thesis. Given an initial and an ending configuration, and some static convex polyhedral model, one can construct a convex polyhedron that fits around the beginning pose and the ending pose. This convex polyhedron i.e. the approximate swept volume can be defined as the minimal convex hull containing both the beginning and the ending configuration (see Figure 7).

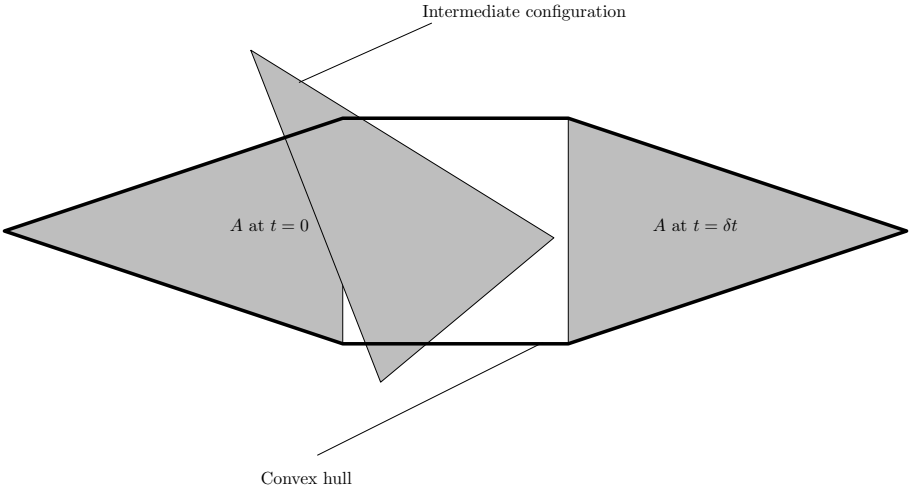

Figure 7: An approximate swept volume in the plane.

When given two pose pairs for two objects, each containing a beginning and an ending pose, and two convex polyhedra, we can construct two approximate swept volumes that both are represented by a convex polyhedron. By using static collision detection queries we can determine whether these approximate swept volumes are intersecting or non-intersecting.

The convex polyhedron that is fitted around the beginning pose and ending pose is non-approximate when linear motion is considered. When the object is rotating (and translating), the swept volume is approximate. Actually, what is likely happen is that the model cannot be physically transported from the beginning pose to the ending pose while not leaving the approximate swept volume. So using the approximate swept volume technique may return false negatives (see Figure 7).

Using exact swept volume collision detection is somewhat harder. The swept volume of a sphere that has an off-center rotation is non trivial to compute. It is even difficult to choose the right representation for this kind of geometry.

When an approximate swept volume query can be computed, it is also possible to call these using binary bisection, giving us a way to approximate the "approximate" time of contact. The binary bisection method is applied on intervals in Section 5.1. With little alteration, this gives us a binary bisection on the approximate swept-volume method. Note that the concept approximation of approximation is used. This might look like the construction is wrong, however, this construction is no different from other constructions that are commonly used within discrete and continuous collision collision detection.

Although this method is non-continuous in the strict sense, it is simple and thus, most likely robust. If discrete collision detection fails, one might want to use this, before attempting to use a method that is fully continuous.

3.3.5 GJK-Raycast

In [23], Gino van den Bergen presents an algorithm to compute the hit point between a ray and a static generic convex object. The algorithm partially carries the name GJK, or Gilbert-Johnson-Keerthi, because it is inspired by an algorithm devised by those three people, i.e. the GJK algorithm. The GJK algorithm computes the positive distance between a convex polyhedron and the origin of the coordinate system. Alternatively, the GJK algorithm can be tailored to return the closest point pair between two convex objects due to the usage of translational configuration space. Translational configuration space is a space that is defined by the usage of Minkowski addition. It will not be discussed here, but for an elaborate description of the GJK algorithm see [22].

Using the same conceptual leap as done with GJK, the GJK-raycast algorithm is capable of determining the time of contact of two colliding (convex) objects under linear motion. It can thus be seen as a good alternative for 4D intersection testing. The GJK-raycast algorithm is capable of approximating the time of contact of a wide class of convex shapes. Both the GJK algorithm and the GJK-raycast algorithm rely on an implicit definition of the convex geometry. When given an explicit definition i.e. the convex hull of a vertex cloud, a corresponding implicit definition is easily found. However, the GJK and GJK-raycast algorithms are also capable of handling a wide class of curved geometry.

3.4 Bounding Volume Hierarchies

Bounding Volume Hierarchies refer to a set of data-structures that allow for the spatial bounding of geometry. Bounding Volume Hierarchies or BVHs can be applied in any number of dimensions. Why do we want to bound geometry? Let us consider a simple, though very complex example. Suppose that we have two Mandelbrot sets that lie in the plane. Both of them have a certain pose within the plane. A Mandelbrot set happens to be conveniently bounded by a disk of radius 2. It may be quite difficult to determine whether two Mandelbrot sets are intersecting. However, if the disks in which the translated and rotated Mandelbrot sets are located do not intersect, then we know that the translated and rotated Mandelbrot sets are also disjoint. So by using a simple (constant time) disk/disk test, we can avoid having to compute something extremely difficult.

The example above explains what a bounding volume is. It still does not explain the "hierarchy" part of the name. It does not make much sense to use the Mandelbrot set to explain this. The latter statement may not be entirely

true, but for the sake of clarity, let us use something less difficult. It should be noted that in this section bounding volume hierarchies are explained within the context of rigid-body collision detection. This means that there are two bodies instead of one.

Before two objects can be explained, let us pick one object first (i.e. a point-cloud). Also let the number of dimensions be 1 and let the number of children of each node in the hierarchy be 2. The objects that are put in the BVH tree are real numbers i.e. 1D-points. Now we end up with something familiar, namely a binary search tree. As is known from binary search trees, a balanced binary search tree has a depth of $\mathcal{O}(\log n)$. Thus by using $\mathcal{O}(\log n)$ comparisons, we can find a leaf (when given a 1D-point). The higher dimensional variant of a binary search tree can be a K-d tree. With a K-d tree the nodes of the tree become axis-oriented planes. Instead of excluding part of the real axis, and thereby excluding a branch and ultimately a lot of leaves, the planes can exclude part of the the search volume in multiple dimensions. To make this explicit, a point is always located in front, on, or behind a plane. Usually one groups on and behind in the same category. Whether a point is in front or on/behind a plane can be determined with a simple constant time test. In the ideal case, with each point plane test, one will exclude about 50% of the geometry.

If the multi-dimensional K-d tree is balanced, we can locate leaves in $\mathcal{O}(\log n)$ time. It is also possible to preform interval queries on K-d trees. If we consider our geometry to just be a set of 3D-points, it is possible to just query e.g. a 3D-interval on this tree. An interval query will be slower than a point query. This is because it may be required to traverse multiple branches of the K-d tree. A 3D-interval, also known as an axis aligned bounding box, can contain geometry. Suppose that we want to statically collide (i.e. intersect) a simple solid model with our point cloud, then we could construct an axis aligned bounding box around our simple model. Supply it to the K-d tree as a 3D-interval. And test the 3D points that are returned for inclusion in our simple solid model. Thus we avoid testing the entire point cloud and this saves us a lot of time.

The problem becomes a bit more complicated than this because of at least three reasons:

1. We are dealing with triangles instead of point-clouds.
2. Both models can be complicated.
3. We have to preform continuous collision detection instead of static collision detection.

Because we are dealing with triangles, the problem becomes slightly more complicated. Suppose that a plane of a K-d tree wants to bound part of a triangular 3D-mesh. In the case of a triangle intersecting the plane, the plane has to be shifted in order to make this possible. Alternatively the triangle can be cut in two parts. The latter has less preference, but it seems unavoidable when dealing with real triangular 3D-meshes. Instead of dealing with triangular meshes, we are now dealing with polygon-meshes. When you cut a triangle into two parts, you are likely to get one quad and one triangle. When the quad is cut again, you may get a triangle and a pentagon. We can also look at this on the bright side, at least the polygons are convex.

Reason 2 makes the problem more complicated than simply traversing a single tree. Two trees need to be traversed simultaneously. How this is done within the context of this thesis is described in Section 5.4.3.

While reading this text, one might think that a K-d tree structure was used within the context of this thesis. This is not the case. The K-d tree was used in order to explain the concept of bounding volume hierarchies. Instead of using K-d trees, a more flexible structure has been used.

How to account for reason 3 is will become clear when reading paragraph 5.1.2. Four-dimensional bounding volume hierarchies are not used within the context of this thesis.

3.4.1 R-Trees

The definition of an R-tree can be brief. It is basically an axis-oriented bounding box tree. Each node in the tree is a bounding box. Research has shown that using axis-aligned bounding boxes as a bounding volume can be extremely fast.

An R-tree can have any number of children at each node. It is even possible to use a varying number of children.

3.4.2 ...-Trees

It is possible to vary the concept of an R-tree. It is possible to use spheres at each node. It is also possible to use axis aligned grid-spaced cubes or rectangles in order to create an octree/quadtree. Next to this one can think of non-axis aligned half-spaces; the so called binary-space-partitions. Actually, it is quite okay to use ones imagination to construct an efficient bounding volume hierarchy for a specific purpose. It is quite difficult to make claims about things like the time complexity of a point query. For example when considering a binary-space-partition, the height of this BSP-tree does not have to be $\mathcal{O}(\log n)$ (this is made explicit in [10]). In fact, when constructing a bounding volume hierarchy, the ideal tree height is $\mathcal{O}(\log n)$. This is often not the case. It should be noted that there exist structures that can do better than $\mathcal{O}(\log n)$, but this concerns certain spatial hashing structures (see [17]). These structures have some assumptions on the geometry.

4 Related work

As said in the introduction, the encapsulating topic of collision detection is simulation. This topic is very broad, because there are a lot of reasonably exotic constructs that are indirectly related to this topic. Because of this, only a small part of the "simulation" topic will be discussed namely the part centered around continuous collision detection.

4.1 Early contributions to Continuous Collision Detection

In [6] (1984), John Canny makes an abstraction of motion and reduces the problem of continuous collision detection to higher order polynomial root finding. The method is based on feature testing, just like the method that is proposed in this thesis (Chapter 5). The solution presented in [6] is made possible trough applying some simplified type of quaternion interpolation, instead of applying quaternion SLERP. This makes the type of motion deviate from linear translational and linear rotational motion (as presented in Chapter 3). The motion of Canny is approximately identical to linear translational and linear rotational motion, especially when amount of rotation per time-step is small. The derivation that is made by Canny is quite elegant and because the problem is reduced to polynomial root finding, the roots can be found using a rather limited amount of numerics.

In [5] (1990), Stephen Cameron presents the concept of 4D-intersection testing. This concept is illustrated in Chapter 3. The publication of Cameron is limited to linear motion (i.e. no rotations).

In [15] (1996), Brian Mirtich presents the concept of conservative advancement. This technique is described in Chapter 3. It is commonly used to preform continuous collision detection. The method of Mirtich is limited to convex shapes. However, recently this method has been extended by others in order to cope with non-convex shapes.

4.2 State of the art methods

In [18], Redon et al. present a continuous method for colliding bodies undergoing articulated motion. The motion abstraction that was used has not been mentioned before in this thesis, but it is called screw-motion. The naming of the motion-type is very appropriate since the motion represents the motion of a screw that is being rotated by a screw-driver. Presumably, the abstracted motion has been used in order to improve the computational efficiency. They use interval arithmetic as a method to find roots of features. In [19], Redon et al. use interpolated motion instead of screw-motion.

In [25], Xinyu Zhang, Stephane Redon, Minkyoung Lee and Young J. Kim constructed a continuous system for articulated bodies that seems to be quite efficient. The system works on bodies consisting out of convex components (thus capable of dealing with non-convex geometry). Conservative advancement is used in order to collide these bodies. The motion-type that is used is the same as used in [19] (i.e. interpolated motion). What is important about their system is that they bound the swept volume of objects by using Taylor models. Taylor models are function inclusion methods and they are described in Appendix E. They happen to be used within the context of this thesis.

In [24], Xinyu Zhang, Minkyoung Lee and Young J. Kim devised a system that is called FAST. It operates on polyhedral shapes. The class of polyhedral shapes is a subset of the class "polygon soups". FAST works by interpolating the motion in-between two poses, like the other methods do.

The follow-up of FAST seems to be Controlled Conservative Advancement [20], which has been devised by Min Tang, Young J. Kim and Dinesh Manocha. The approach seems reasonably similar to that of FAST, although the main difference is that it operates on polygon soups. Like FAST, they also use interpolated motion. More on this method will be written in Chapter 5, because Controlled Conservative Advancement will be used to benchmark the method devised within the context of this project.

4.3 Bounding Volume Hierarchies

For the context of this project, the main reference to BVHs has been an excerpt of chapter 1 from [12]. Here Herman Haverkort gives an overview of the concept of Bounding Volume Hierarchies. In [12], asymptotic bounds are given that describe the query time of certain bounding volume hierarchies. Making claims about the asymptotic query time of bounding volume hierarchies seems to be rather difficult.

In [21] the advantages of using axis aligned bounding box trees are discussed. Within the context of this project, axis aligned bounding boxes are used in a similar, but non-equivalent manner. The content of [21] (which is also in the PhD thesis of Gino van den Bergen), has been found at the near end of this master project. If this paper were to be read earlier, this would have most likely influenced the construction of the bounding volume hierarchy used within this project. The main difference between the bounding volume hierarchy that is presented in Section 5.4.1 and the approach of van den Bergen is that van den Bergen does not cut the geometry.

Part of the approach used in this thesis has been discussed with Herman Haverkort. Haverkort's thesis partly focusses on the data structures whereas the thesis of van den Bergen partly focusses on applying these structures within the context of collision detection.

4.4 Physical simulation

The way motion should be handled is similar to the way David Baraff describes it in [2]. The paper primarily deals with what should happen after a collision has occurred. In [2], Baraff abstracts from the asymptotic precision of the motion. This is doable, since the collision detection method used is discrete. Additionally binary bisection is used to approximately find the time of contact.

In [7], Erin Catto presents an iterative algorithm that is able to compute constraint forces between rigid objects. The constraint forces are determined by an algorithm that is based on projected Gauss-Seidel. A simplified friction model is used. The algorithm runs in linear time, linear with respect to the number of constraints. Catto does not consider constraint impulses explicitly. In [4], impulse constraints are described. In order to construct a rigid body simulation, you do need impulse based collision response.

5 Proposed Continuous Method

The method proposed in this section is based on Feature-Testing, Interval-Arithmetic/Taylor-Lagrange based inclusion functions, Taylor-Lagrange based root-finding and Bounding-Volume-Hierarchies.

Constructing a continuous collision detection system based on interval arithmetic seems to be a conservative choice. Interval-arithmetic is a logic based construct with which one is able to prove certain properties of functions. For example: "Function f does not contain a root in domain D.". Such facts can be used in order to make safe assertions about intersections of certain features.

When using real numbers, we have to deal with approximations of real numbers called floating point numbers. Collision detection systems based on interval arithmetic, in particular root-finders based on interval arithmetic seem quite capable of doing this. They are stable in the sense that a reliable result is computed every time such a root finder is used. Within the context of this thesis interval arithmetic is used in a way that does not account for floating point rounding. Thus strictly speaking, facts derived with interval arithmetic within the context of this project should not always be taken as absolute facts.

It is difficult to label root-finding based on interval-arithmetic as being "slow". The word slow can be used in different contexts. Within the context of this thesis, one of the goals is maximize computational efficiency.

5.1 The 2D case

Before reading this section, it is required that the reader is familiar with the background section (especially Section 3.3.1).

5.1.1 Brute-Force 2D Continuous Collision Detection

In 2D, within the context of this project, an object is defined as a set of edges. The edges are defined locally in object space. Imagine two of these sets, defining two objects, each having a certain pose. Of course these two objects are animated. The animation of a single rigid object can be seen as a time-dependent transformation from local object space, to world space. This 2D transformation is described in Appendix A.

An edge is an ordered list of two points. We thus have two kinds of objects: point and line-segment. If and when two objects have collided at $t = TOC$, there can be one or multiple contact points. The method proposed here is capable of retuning one of these contact points (and associated contact normal).

With 2D continuous collision detection, one can make the observation that when two objects are in contact. It is always the case that a vertex of object A is touching a line segment of object B or that a vertex of object B is touching a line segment of object A.

The property above is fully exploited in order to find the TOC in a brute force manner. For each vertex of object A one has to test whether it will hit any of the edges of object B and vice versa. If one wishes to determine the TOC, then one simply picks the minimum TOC of all feature/feature combinations (see Figure 8).

Note that in Figure 8, the shapes are convex. With this method the shapes are not required to be convex.

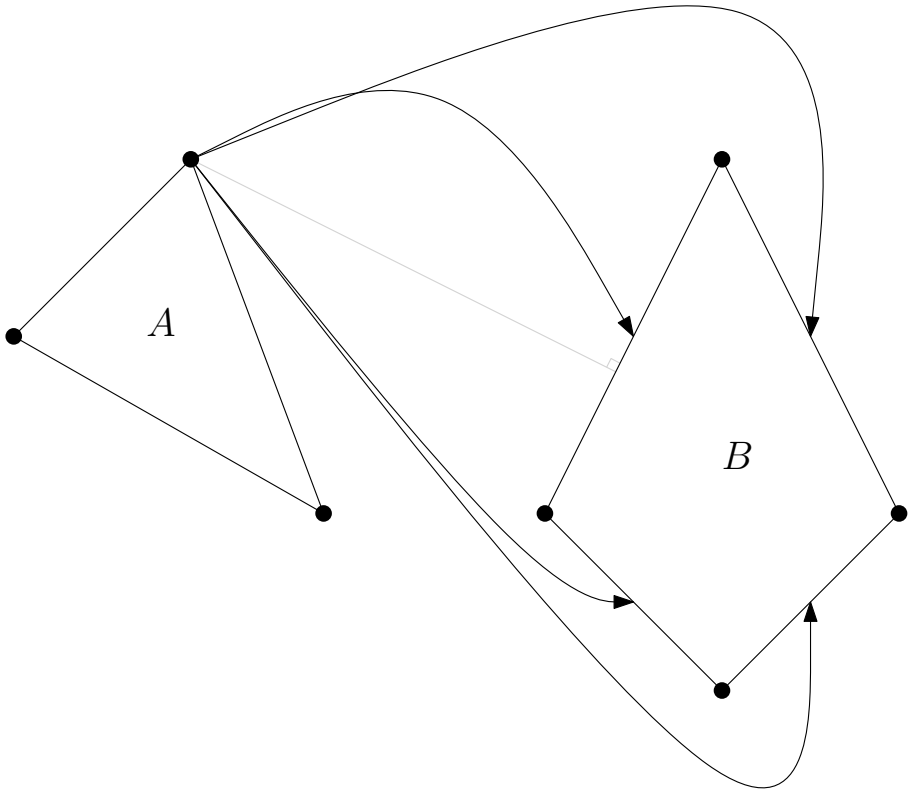

Figure 8: Brute forcing all combinations.

In order to be able to compute the TOC, one must also be able to compute the TOC of an animated point and an animated line segment. Within the context of this project, in the 2D case, this is done with interval arithmetic.

5.1.2 Interval Arithmetic based Root-Finding

As described in Section 3.3.1, in order to devise a continuous test, one has to construct a distance function. This distance function can be very complicated, and that is why it is very difficult to use analytical techniques to solve the roots. In Section 3.3.1, a motivation is given to use interval arithmetic in order to solve the roots of such a distance function.

The distance function we have to solve denotes the distance between a point and an line (both subject to linear translational and linear rotational motion). The motion type is listed between parenthesis, because it is possible to abstract from the type of motion. A moving line is defined by two (different) parametrized point trajectories, and a moving point is defined by a single parametrized point trajectory. It is possible to derive an optimized parametrized distance function for a specific kind of motion, or it is possible to supply point-trajectory functions to the distance function in order to compute the distance dependent on time while leaving the motion type customizable.

The latter is done within the 2D phase of this thesis. By simply defining the motion of a point as is done in Appendix A, it is possible to sample distances

at certain moments in time as done in Appendix B.2.1. Note that in Appendix B.2.1, also two time derivatives of this distance function are derived. Within the context of this project, in the 2D phase, no time derivatives were used in order to find the roots of points and lines. The time derivatives can be used to optimize the root finding though. More on this will be written in Section 5.2.

For the reader that is totally unfamiliar with interval arithmetic, the basic concept of interval arithmetic is described in Appendix D.

It is possible to construct an interval-arithmetic based root finder that takes a distance function, a time domain and an time epsilon value as input. The function will return a small domain r, $r_{\text{upper}} - r_{\text{lower}} < \epsilon_t$ that is likely to contain the first root of the distance function within the domain.

As said in Section 3.3.1, not all roots are considered to be intersections. After a root has been reported the root has to be tested for inclusion within the line segment. Only then, the root can be reported as a real contact between a point and a line-segment.

In 2D is it quite doable to make a conceptual simplification to the root finder. Instead of supplying a distance function, a domain and an epsilon, it is sufficient to supply a boolean function, a domain and an epsilon. As you may have guessed, the boolean function is there to indicate intersection.

Let the boolean function be defined as follows: $b : \mathbb{IR} \mapsto \mathbb{B}$. When given a time domain, the boolean function indicates whether the the two features could be intersecting. As the span of the input domain of the function converges to 0, the boolean function becomes exact. Meaning that when the input domain is larger than 0, the function may return false-positives.

There is quite some rationale behind this, if the input time domain span is 0, then the function is just the result of a static collision query. As the domain gets larger (i.e. non-zero), the boolean function can be thought of as a static collision detection query on two swept volumes (allowing for false positives).

Algorithm *FindFirstRoot*(b, d, ϵ_t)
1. $f \leftarrow d_{\text{upper}}$
2. **while true**
3. $\quad q \leftarrow b(d)$
4. $\quad$ **if** q
5. $\quad\quad$ **then if** $d_{\text{upper}} - d_{\text{lower}} < \epsilon_t$
6. $\quad\quad\quad$ **then return** d
7. $\quad\quad\quad d \leftarrow [d_{\text{lower}}, \frac{d_{\text{lower}}+d_{\text{upper}}}{2}]$
8. $\quad\quad$ **else**
9. $\quad\quad\quad$ **if** $d_{\text{upper}} = f$
10. $\quad\quad\quad\quad$ **then return nil**
11. $\quad\quad\quad d \leftarrow [d_{\text{upper}}, f]$

Floating point issues regarding this root finder

The parameter ϵ_t may get arbitrarily small. Actually ϵ_t may even reach 0. In the latter case, the code that is listed above does not work. It is actually already possible that the code above fails when using reasonably small epsilon values. This is due to the fact that the floating point expression $m = \frac{d_{\text{lower}}+d_{\text{upper}}}{2}$ does not always obey $d_{\text{lower}} < m < d_{\text{upper}}$. In fact, it may be the case that $m = d_{\text{upper}}$, in this case, the search domain will not converge.

In order to cope with this, some additional code has been written to select a floating point value m that does obey $d_{\text{lower}} < m < d_{\text{upper}}$. It may be the case that such an m does not exist, i.e. when d_{upper} is the successive value of d_{lower}. In this case, it is no use to iterate further to gain more precision. Because we are in a case where q holds, it is allowed to simply return d as being a very small domain.

A boolean function

For those who are interested in to see how the edge-point boolean function looks like, the code that was used within the 2D project is listed in Appendix H. Note that this example does not use the derivations made in Appendix B.2.1. The construct differs from Appendix B.2.1 in the sense that a transformation is made such that one object is fixed (see Figure 9). In Appendix B.2.1, the problem is solved in world space. The advantage of working in world-space is that it doesn't cause much confusion when dealing with derivatives.

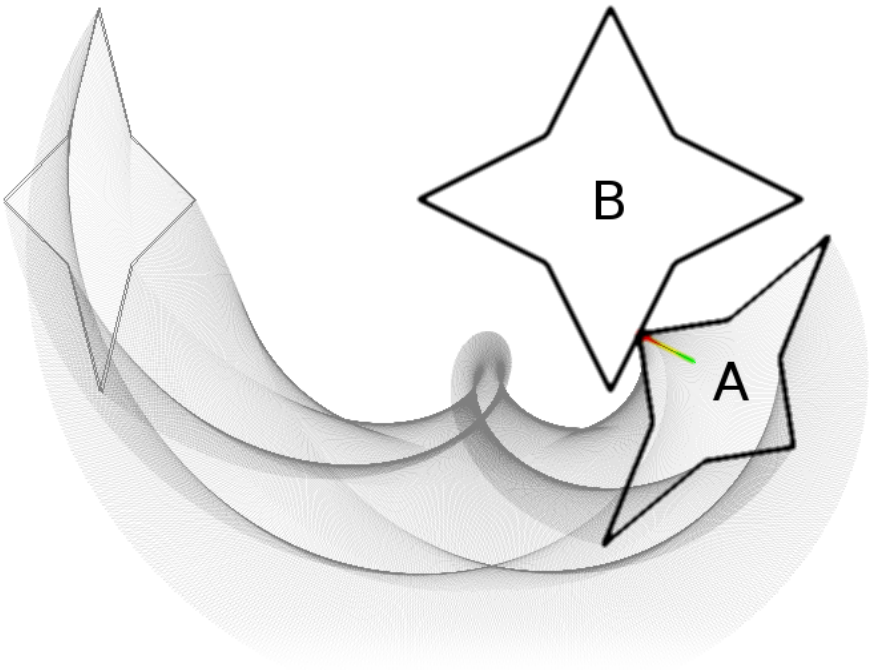

Figure 9: Two objects under linear translational and linear rotational motion, viewed from object B.

As can be seen in Figure 9 and in Figure 2, the contact normal is easily computed once the two features that cause the contact are known.

5.2 The 3D case

In 3D things are a bit different. Instead of having edges and points, now we have points and convex polygons. Because of the topology of a model, it seems to be rather in-efficient to just test all polygon/polygon combinations. This is because polygons share vertices and edges.

The convex polygon can be decomposed into the following three components:

- point
- line
- plane

If we take all distinct sets of size two from the set above we get a set of cardinality 6:

- {point, point}
- {point, line}
- {point, plane}
- {line, line}
- {line, plane}
- {plane, plane}

Now we have to prune the combinations that do not make sense. Can a point collide with a point, in an artificial setting it is possible. However, leaving out these kind of collisions does not seem to hurt. In 3D, a point/line collision can be also be considered to be artificial. A point and a plane can definitely collide, and following the same criteria, a line and a line can also collide. A plane is considered to have an unbounded surface. Whether two planes intersect is a matter of them being parallel or non-parallel. This is non interesting in the context of this application. Likewise the TOC between a line and a plane also does not make much sense[2].

Two types of computations will remain:

- point/plane TOC query
- line/line TOC query

Note that the remaining computations apply on lines and planes instead of on edges and polygons. What we really would like to know is the TOC between a point and a convex polygon or the TOC between two edges. This result is obtained in a similar way as done in the 2D case. First the TOC of a point and a plane is computed. When this result is known, a static test is done to determine whether the TOC is also the TOC of a point and a polygon associated with the plane in question. The same applies for line/line tests. First the TOC is computed of two touching lines. Later, a static test is preformed to see if this TOC is indeed the TOC of two intersecting edges.

[2]When the line is a parametrized equation, then one can solve for a time of contact. This is a different application though.

There are multiple ways to solve the line/line intersection problem. One way is to assume that the lines are co-planar, and then solve the problem in 2D. The way this problem was solved within the project associated with this thesis is that the line segments are enlarged because there is uncertainty about their ending points. Once the lines are enlarged, one can use rectangle/rectangle tests, rectangle/disk tests, and disk/disk tests, see Appendix C.

Again, distance functions are used in order to compute the distance between the features. To be complete, the word distance can be omitted. Because we are only interested in the roots of such a function, it does not matter much if the function actually represents the distance between the features. So we abstract a bit further, and construct functions of which the roots are of interest. In Appendix B.2.4, the distance function between a point and a plane is derived. The point and the plane can have a customizable type of motion.

In Appendix B.2.3 a function is constructed that yields a scalar for any configuration of two lines. Whenever this scalar is 0, then the lines are either intersecting, or they are parallel. Also this derivation abstracts from the type of motion.

There exists the notion of plane space. I.e. a coordinate system that is uniquely defined by a plane in e.g. the space of object B. By using homogeneous 4×4 matrices, it is not that difficult to construct a transformation that transforms a point in the space of object A to the plane space of object B. If a point is in plane space, then the $z-$coordinate simply measures the Euclidean distance to the plane. The $xy-$plane in plane space represents the plane itself (after being rotated and translated). When raising the latter transformation to an interval arithmetic transformation (instead of taking a time parameter it takes a time interval parameter, see Appendix D), we can transform a point in A-space, to plane space. The result will be an axis aligned beam in plane space. If this beam strictly does not intersect the $xy-$plane then we know that the point does not hit the plane. If it does, then we can preform a test to check whether the beam hits the polygon located on the $xy-$plane.

By using the basic root finder presented in Section 5.1.2, the TOC between a point and a plane both under customizable motion can be approximated.

When using interval arithmetic, the bottle-neck definitely lies in interval-multiplication. Interval-multiplication is very slow. This is because lots of "if" statements are used in order to preform it.

Eventually in 3D the constructs in Appendix B.2.4 and B.2.3 were fully exploited (including the derivatives). Before explaining the way the derivatives were used, let us first explain why interval arithmetic alone is not adequate enough.

Diverging computations

With interval arithmetic, a problem that will arise is that the computation may diverge. Suppose that we have a large chain of computations. This can be represented with a function ($f : \mathbb{IR} \mapsto \mathbb{IR}$). If t is an interval with span 0, interval arithmetic requires that the function results into a single value. Now for example, assuming that the computation inside $f(t)$ is elaborate, we can take an tiny interval of e.g. a span of 2^{-16}. This can actually result in an output range having a span of 1. If this happens to be the distance between a point and a plane, than obviously it is not sufficient to use a time epsilon of 2^{-16}. In

order to make it work, the time epsilon supplied to the root finder has to be made very small. So small that it is less worthy of being called efficient. One would expect that making the input twice as small would cause the output to also become twice as small. Visual inspection of colliding object shows that this is not the case. One might expect that there are more simplistic ways to show this. Indeed there are, however this approach has not been investigated.

To conclude the above, the initial 3D results using interval arithmetic were disappointing. The collisions were only satisfying when using a very small epsilon. In this case, the collision rate would drop below an acceptable rate. The slow performance is most likely caused by the slow convergence of the interval arithmetic based root finder.

5.2.1 A root finder based on Taylor's theorem with the Lagrange remainder

The initial design of this root finder is presented in Appendix G. The method uses the "distance" function itself, its first time derivative and its second time derivative over an interval in order to compute two parabolas that bound the function (see Figure 10). The function is sampled at a certain moment in time. Likewise its first time derivative is sampled. Next we determine a span in which we want the bounding parabolas to be valid. We sample the second time derivative over this time span. When sampling a second time derivative over a time span, we get an upper bound and a lower bound for this value. The upper bound determines the curvature of the upper bounding parabola and the lower bound determines the curvature of the lower bounding parabola. It is guaranteed that the function is bounded by the upper and lower parabolas (in order to get an understanding of this see Appendix I).

The green lines represent interval arithmetic bounds on the 0^{th} derivative of the function (i.e. the function itself). Likewise the blue lines represent bounds on the function that are determined by the parabolas. As one can see, the blue (parabola-) bounds are not always tighter than interval arithmetic bounds. They do get tighter as the size of the search domain decreases.

If the parabolas do not hit the horizontal axis within the time domain in which they are valid, then the inner curve will also not hit the horizontal axis within this time domain. If both parabolas intersect the horizontal axis ($t = t_6$, Figure 10) within the valid domain, then it is certain that the inner curve will intersect the horizontal axis within that domain. These mathematical conjectures are implied by the construct and can be used in order to speed up the root finding that is done with plain interval arithmetic. A mixture of the method presented in Appendix G and the interval arithmetic based root finder presented in Section 5.1 has been used to solve roots within the context of 3D continuous collision detection.

There exists a class of function inclusion methods called Taylor models. These are explained in Appendix E. They differ from the function inclusion method presented above in the sense that the vertical distance between the lower bounding polynomial and the upper bounding polynomial is always constant. In fact, a Taylor model is just a polynomial with vertical thickness making it able to bound complicated functions.

The advantage of two bounding parabolas above Taylor-models is that the thickness of the two bounding parabolas is 0 at the point where the function

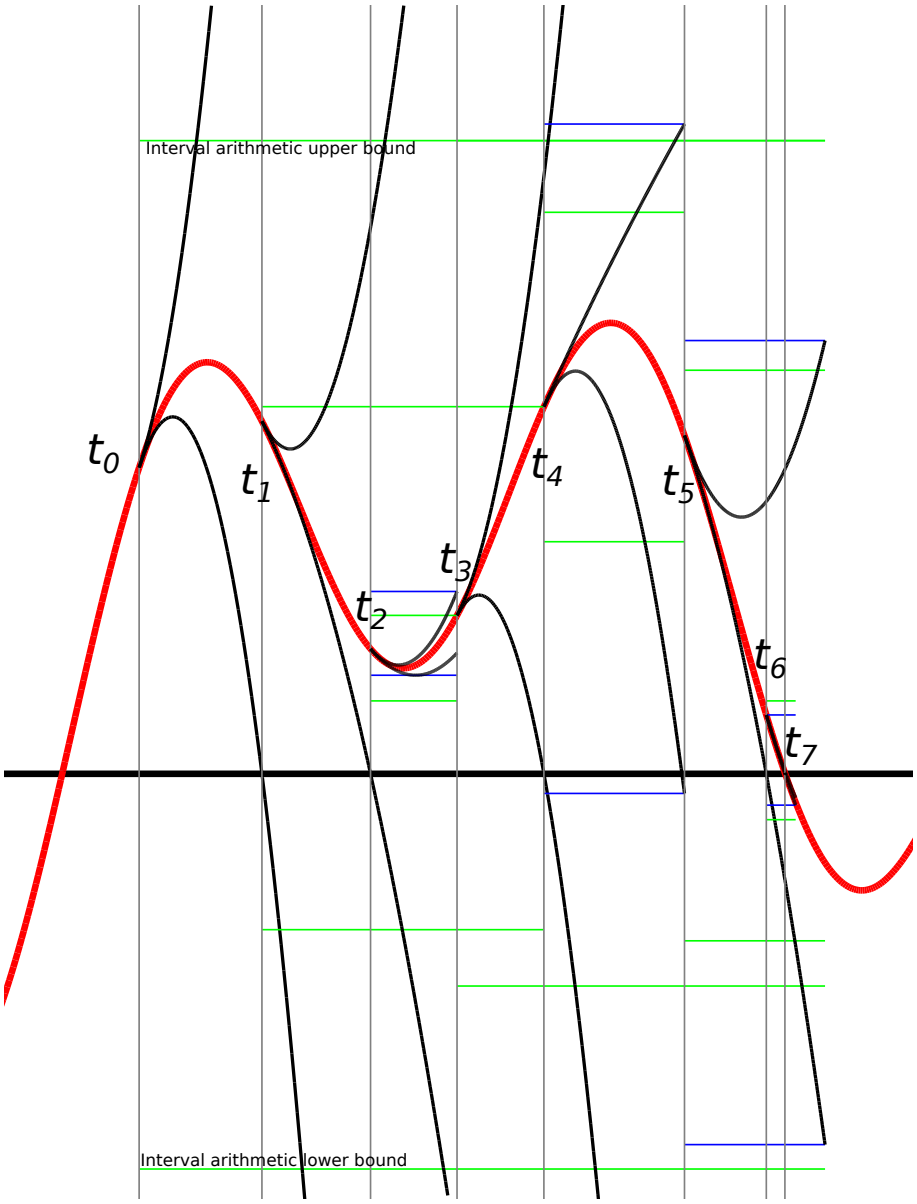

Figure 10: The operation of the root finder.

and its time derivative are sampled. When computing the roots of the two parabolas, one will always get some advancement (see Appendix G). This need not be the case with Taylor models.

However, two bounding parabolas are conceptually more difficult to handle than a single polynomial that is vertically fattened. It might be possible to construct a root finder based on first order Taylor models that also uses binary bisection that may be faster than the root finder constructed within this thesis. When using two parabolas, one does not require bisection. Using both bisection and bounding parabolas causes the root finder to converge faster though.

There exists one root finder that is similar to the root finder presented in Appendix G and that is the Interval Newton method. The Interval Newton method uses an upper bound and a lower bound on the first derivative in order to bound the function curvature. The Interval Newton method has not been investigated within the context of this project. A method based on the Interval Newton method, might be capable of quickly finding the first root when only one derivative is available.

However, a lot of thought has been put into this. Using the Interval-Newton method, does not seem to be a right choice because the curvature of the functions is not well approximated by a diverging slab (as is the case with the Interval Newton method). Using the Interval Newton method will still remain an option because it has not been tested.

Algorithm *FindFirstRootHybrid*($f, \frac{df}{dt}, \frac{d^2f}{dt^2}, d, \epsilon_t$)
1. $f \leftarrow d_{\text{upper}}$
2. **while true**
3. q $\leftarrow$*IntersectingAtRoot*(d)

($*$ Possible intersection $\equiv q$ $*$)
4. **if** q
5. **then** $g_{\text{domain}} \leftarrow f(d)$
6. $g \leftarrow f(d_{\text{lower}})$
7. $\frac{dg}{dt} \leftarrow \frac{df}{dt}(d_{\text{lower}})$
8. $\frac{d^2g}{dt^2}_{\text{domain}} \leftarrow \frac{d^2f}{dt^2}(d)$
9. Construct a second order inclusion function i.e. two parabolas based on g, $\frac{dg}{dt}$ and $\frac{d^2g}{dt^2}_{\text{domain}}$.
10. $q \leftarrow (0 \in g_{\text{domain}})$**and**$(0 \in$ *VerticalSpan*$(g, \frac{dg}{dt}, \frac{d^2g}{dt^2}_{\text{domain}}, d))$
11. **if** q
12. **then if** the inclusion function does not intersect the horizontal axis within d
13. **then** $q \leftarrow false$
14. **else**
15. **if** the inclusion function intersects part of the horizontal axis within d
16. **then** adjust d to match the (first) intersection (within the original d)
17. **if** q

($*$ Domain d is likely to have been adjusted. $*$)
18. **then** $q \leftarrow$ *IntersectingAtRoot*(d)
19.
20. **if** q
21. **then if** $d_{\text{upper}} - d_{\text{lower}} < \epsilon_t$
22. **then return** d
23. $d \leftarrow [d_{\text{lower}}, \frac{d_{\text{lower}}+d_{\text{upper}}}{2}]$
24. **else**
25. **if** $d_{\text{upper}} = f$
26. **then return nil**
27. $d \leftarrow [d_{\text{upper}}, f]$

It must be noted that the algorithm is more or less similar to the algorithm

presented in Section 5.1. The final lines of the algorithm, starting with line 20 are actually identical.

At line 5, a regular interval arithmetic query is preformed on the function itself. This interval is converted to a boolean in line 10. If the boolean is true, this implies that there may be an intersection. Lines 11 to 16 are added in order to prune this possible intersection. First a check is done if the second order inclusion function that was constructed, can be used to assert that there is no intersection. If so, q is basically set to false and we preform a normal iteration of the interval arithmetic based root finder. Note that just using this construct already gives a considerable speed-up. However, when the inclusion function is hitting the axis, the parts that do not hit the axis can be excluded. Recall that in general a parabola may have two roots. Thus our second order inclusion function may intersect, leave the axis, and then intersect again. However, it is still possible to make safe bounds on where roots may or may not occur. One can construct a lower bound for which it is known that the first root cannot occur before that. If both parabolas are intersecting the horizontal axis within the domain, one can also give an upper bound (see Appendix G). If we get an upper bound and a lower bound, then the convergence of the root finder seems to be very fast.

At lines 3 and 18, the function *IntersectingAtRoot* : $\mathbb{IR} \mapsto \mathbb{B}$ indicates whether there is a possible intersection within the specified time domain.

The content of the function is specific to the type of query that is being made. It can either be a weakened point/polygon query. Or it can be a weakened edge/edge query. In both cases, the assumption that everything is within the plane, reduces both the edge/edge intersection and the point/polygon intersection, into two 2D problems.

Instead of first finding roots and afterwards pruning these roots, a different approach has been taken to increase performance. The detection of roots and the pruning of roots happens simultaneously. This seems to be an efficient configuration.

Note that *IntersectingAtRoot* does not require any derivatives to be used. The function name ends with "AtRoot" because one is allowed to make an assumption that the features are co-planar. Note that *IntersectingAtRoot*, considers a scene over a span of time, not at an instant. A moving vertex can thus be seen as a axis-aligned bounding-box that contains the trajectory of that vertex. When we consider two line segments, we are actually considering four moving vertices. Deciding when two fattened edges intersect is not trivial. In Appendix C, the case of four moving vertices is simplified into a 2D test.

Whether a moving point will hit a polygon, can be computed quite efficiently by fixing the polygon into the $xy-$plane. The point trajectory will be represented by a beam. If the cube's $z-$coordinate spans through the $xy-$plane, there may be an intersection. Next to this, the rectangle (as viewed through the $xy-$plane, has to intersect with the polygon. The latter is done with the help of cached-half-spaces. For each edge of the polygon a half-space is precomputed. One can easily test for the inclusion of a point in the polygon. The problem is that we need a rectangle in polygon test. Thankfully this problem is solved quite easily. The measure for the distance to an edge is a direct computation that only uses addition and multiplication. Replacing the point by a rectangle in $\mathbb{IR}^2$, fixes the problem. Instead of yielding a scalar, representing the distance to the line of a line segment, an interval is produced that bounds the previously

mentioned quantity.

Floating point issues involving the hybrid root finder

This is the point where the root finder becomes less clean. The interval arithmetic algorithm presented in Section 5.1 has some floating point issues, but these have been resolved in a neat way. This is not really the case with this root finder. The roots of the parabolas that are used are subject to some error. The way this has been resolved is by subtracting an epsilon from the lower-bound and afterwards clamping it to the original domain d. Likewise the upper bound is incremented with an epsilon and clamped to domain d. This may seem like a good solution, however there is little that safeguards correct operation of the root finder in extreme situations. This is because the epsilon values that were used have been guessed. The epsilon value is dependent on the shape of the parabolas.

In practice the root finder does quite well. But making the epsilon twice as small causes the root finder to miss roots in a very incidental basis. Luckily, where the parabolas fail (in accuracy) the interval arithmetic takes over. Where the parabolas are incapable of determining a finer interval, the interval arithmetic algorithm can keep cutting the domain into parts until the "bottom" of the arithmetic has been reached.

Loss of significance

The roots of a parabola can be computed in a "naive" way using the quadratic formula as is done within the context of this project. However, one may argue that there exist better ways to compute the roots of a parabola. Some effort has been undertaken to account for loss of significance. However, accounting for loss of significance is rather troublesome, due to unfamiliar mathematics and new degeneracies that arise trough doing so. One might even want to consider using a library (written by an expert) to compute the roots of a parabola[3].

Degeneracies

There are lots of degeneracies that may occur when implementing the root finder above. One of them is very awkward, and causes the system to become very slow. The system, as presented above assumes that the function which is supplied incidentally intersects with the horizontal axis. What if the function would be a line that would endure intersection with the horizontal axis over a period of time. One can argue that such a situation would not show up, and in many tests it did not show up. However, when resorting to randomized tests, this degeneracy did show up, causing the system to become extremely slow. It used to be extremely slow because an entire range of roots had to be declared. It is already slow to basically iterate over these roots. Computing them is just a waste of computing power and it serves no purpose.

In the system, enduring roots translate to the following two situations:

1. A vertex moving tangential to a plane (on that plane).

[3]The solution on `http://en.wikipedia.org/wiki/Loss_of_significance` does not seem to suffice, due to degeneracies that are not treated.

2. Two parallel edges that move while remaining parallel.

3. Enduring touch of crossing lines

If a vertex is moving as described in situation 1, then a possible collision does not contribute much to our 3D collision detection system. This is because it is only possible for the vertex to hit the side of a polygon. Because these polygons are "infinitely thin", and because such a collision should be covered by the other features of the model, this type of collision can be pruned.

Situation 2 describes two parallel edges. In such cases these edges can pass trough each other. Because edges are modelled as being "infinitely thin", this is okay.

The last situation (3), can either represent the enduring touch of two lines or the enduring touch of two edges. Let us assume that two edges are in enduring contact. It must be noted that the contact is enduring, i.e. the edges are not passing trough each-other. This is the motivation for also ignoring this case.

How to solve the degeneracy described above has been the subject of experimentation. The situation where the root finder is about to declare a root is recognized, i.e. the function is approximately 0. If the slope of the function is approximately 0 too, then the search domain of the root finder is skipped until the function itself deviates from being approximately 0. The parabolas are used to assert whether the function is approximately 0 over a domain.

The observant reader might have noticed that the declaration $g_{\text{domain}} \leftarrow f(d)$ was used within the hybrid root finding algorithm. This construct can also be used to assert that the function is approximately 0 over a domain. Thus there are two ways to do this. Both the parabolas and the sampled interval bound the range of the function over the domain. The parabolas usually happen to be tighter, but not in all cases. Eventually, a bounding interval was computed based on the parabolas. This interval was then intersected with the sampled interval bound in order to obtain a better bound.

The degeneracy fix has not been incorporated in the algorithm within this section. This is because the pseudo code is a simplification anyway.

Is this faster than Interval Arithmetic?

The answer to this question is yes. If the root finder is replaced by an interval arithmetic solver, and when the first and second derivative computations are removed, the system becomes 6 to 9 times slower.

5.3 Making the system Non-Penetrating

In the previous sections it is shown how to get an approximation of the TOC with parametrizable accuracy. This parameter has the name ϵ_t (i.e. time based epsilon). The problem with the conceptual system above is that the objects are touching at the TOC. They can either be disjoint, touching or slightly penetrating. It would be preferable to limit the amount of penetration or disjointness by being able to specify a distance epsilon (ϵ_d).

5.3.1 Limiting the amount of deviation

Time and distance happen to be related by the concept of velocity. It is possible to compute an upper bound for the worst case relative velocity between two objects.

For each pair of leaves, it is possible to give an upper bound for the relative velocity in between. This upper bound can be tight, but it is not required to be tight. Computing this upper bound (a positive scalar) is actually a very cheap computation.

Using this cheap computation, it is possible to compute a lower bound for the time epsilon i.e. $\epsilon_t = \frac{\epsilon_d}{\texttt{relativeVelocityUpperBound}}$.

The root finders presented earlier return a domain instead of a single value as a root. By taking the lower bound of the domain that is retuned, the error is biased towards non-penetration. The lower bound may still penetrate due to arithmetical errors (floating point values are rounded to even meaning that the bounds of the root are not strict bounds).

In the conceptual model, $\epsilon_t = h$ means that that the lower bound of the root that is returned does not deviate more than h time units from the actual root. Within these h time units, a relative distance of at most
$\epsilon_d = h \cdot \texttt{relativeVelocityUpperBound}$ can be covered in any direction.

5.3.2 Take a step back

By considering the above, in most cases the system does not cause penetration. However, due to imprecise rounding of floating point values, this may be the case. In order to cope with this and to make the system strictly non-penetrating, strictly non-penetrating with the exception of possible bugs, we take a step back in order to enforce non-penetration. Taking a step back is easily done by decrementing the TOC and clamping it to the non-negative domain afterwards.

However, we want the two objects to be at least ϵ_d units apart. At the non-decremented TOC, there is a contact point, and an associated contact normal. The relative velocity between the objects can be computed at this point. This vector can be designated with the name "impact velocity". This vector can afterwards be measured along the normal of the contact point (using the dot product). It is thus possible to measure the impact velocity in the direction of the contact normal. We can compute δTOC as being a first order approximation $\delta TOC \cdot \texttt{impactVelocityDotNormal} = \epsilon_d$. Thus $\delta TOC = \frac{\epsilon_d}{\texttt{impactVelocityDotNormal}}$. We can thus state that $TOC = TOC' + \delta TOC$ or $TOC' = TOC - \delta TOC$ (such that TOC' is non-negative).

Note that the approach discussed above is not perfect (see Figure 11). Though in practice very decent results can be obtained by using it see Section 7. The situation has been simplified for illustrative purposes (in general the face is not static). Note that d is the distance between both objects if one object were to consist out of a single vertex. Although in theory the picture may get worse, in practice it does not. The difference between the actual point trajectory and its first order approximation based on the impact velocity is negligible. If the angle of impact has a cosine of 1, then the result is almost perfect. The steeper the angle gets, the bigger the factor in difference between ϵ_d and d may get.

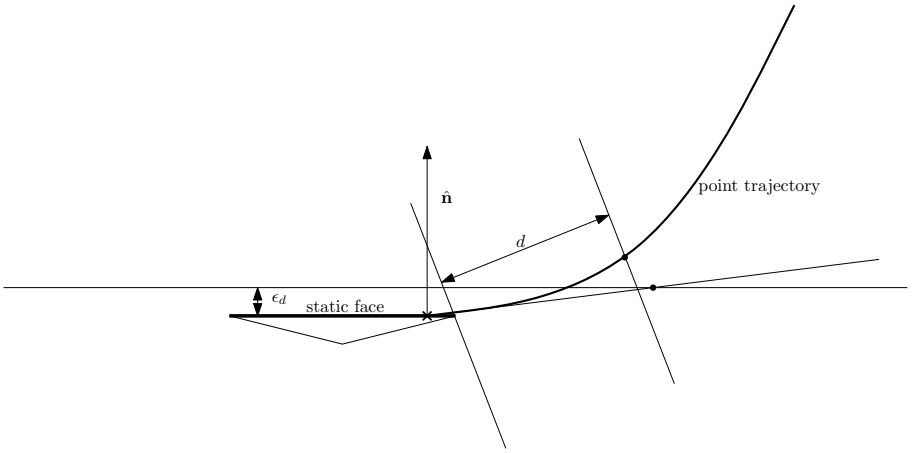

Figure 11: Figure illustrating that the separation distance may be a lot bigger than ϵ_d.

One detail that is important to mention is that the variable `impactVelocityDotNormal` is always positive. If there is a collision, the relative velocity at the point of impact should always be headed towards penetration. If this is not the case than this means that the contact normal has to be flipped and and that the variable `impactVelocityDotNormal` has to be negated, effectively causing the contact normal to have the right orientation and the `impactVelocityDotNormal` to always be positive.

Because the direction of normals on edges is rather ill defined and because faces can collide with their back-sides, postulating that the variable `impactVelocityDotNormal` is always positive seems to always make the contact normals consistent.

5.4 Improving Performance

Up until now, 2D and 3D continuous collisions have been discussed in a brute force manner. This type of collision detection is also called raw collision detection. The approaches above fail when the number of features grows. It would not be sufficient to just state that they are slower, the systems simply become unusable when the number of features exceed a certain (low) threshold. The quadratic complexity of the brute-force algorithm says part of it. It is actually quadratic complexity with a big constant.

The performance has been boosted in two concrete ways:

1. Factoring the algorithm in such a way that the constant of the quadratic term becomes smaller and the constant of the linear term becomes bigger

2. Exploiting the concept of Bounding Volume Hierarchies

As is not clear from the enumeration above, both are quite related. This is because both methods rely on non-precise continuous sphere/sphere tests. Some readers may find that continuous sphere/sphere tests are simple to preform. Within the context of this project, the spheres have an off-center rotation, thus

making a continuous test significantly harder. Above, it is stated that an non-precise test is used. This is because we are not that much interested in when the spheres collide. We are interested in when the geometry collides that is within the spheres. There's little to gain if one is able to approximate the TOC between two "bounding" spheres with an accuracy of e.g $1.0 \cdot 10^{-10}$.

Let us start with discussing the bounding volume hierarchy that was used. The bounding volume hierarchy itself i.e. the data-structure, in 2D is quite similar to the data structure in 3D. Therefore less focus will be given on the number of dimensions.

5.4.1 A custom Bounding-Volume-Hierarchy

When given a complicated model in 2D or in 3D, collision detection becomes infeasible due to the high number of features. By chopping up the model into tiny parts. We try to only test those parts that actually will collide. How do we know which parts of the model will collide? It happens to be simpler to ask which parts will not collide. The latter can usually be answered by a computation of limited complexity for each part.

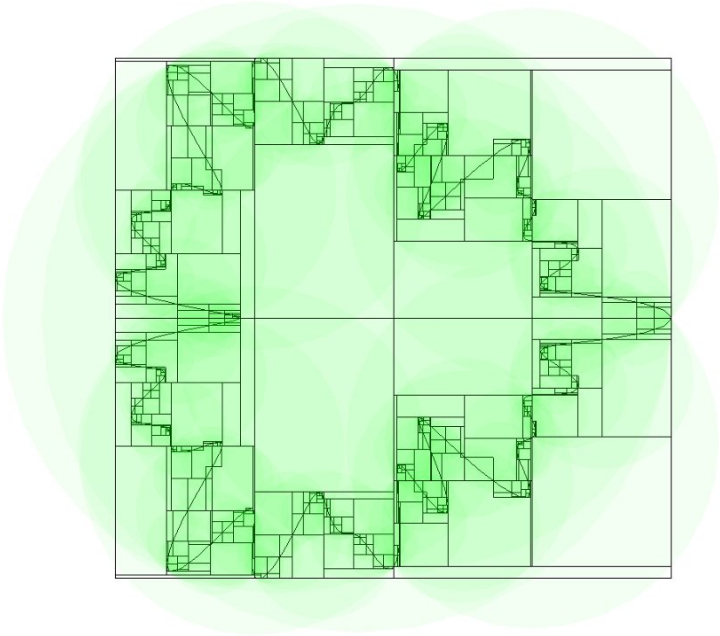

Figure 12: A bounding volume hierarchy. (The model is a parametrized cyclic radial curve.)

As may still be difficult to see from Figure 12; there is a hierarchy of bounding rectangles. The outer rectangle bounds the entire model. One may also be able to see green circular shapes around the geometry. Actually, there is a big disk bounding all the geometry (which may be invisible due to the type of blending that was chosen). The disks that are drawn around the geometry are minimal. If one is able to construct minimal spanning disks for a set of points, one is

capable of constructing minimal spanning disks around edges. In 3D a similar statement holds about spanning spheres and polygons.

Note that finding a minimal spanning disk is a classic geometrical problem. It is described in [10]. Due to the complexity of this problem, one might be persuaded to fit a disk around a bounding rectangle. This will lead to an inferior bounding-volume-hierarchy, and this is not the approach taken within this thesis. Alternatively a 3D minimal spanning sphere algorithm can be derived from [16]. Both the 2D and the 3D algorithm look very similar.

Floating point issues involving the minimal spanning sphere/disk algorithm

Implementing these algorithms is not easy. In fact the implementation done within the context of this project has certain unresolved degeneracies. Whenever such a degeneracy occurs, the algorithm is terminated (by throwing an exception). Because the implementation will only fail in extreme cases, and because the construction of the disks/spheres occurs at start-up, this is not a big issue. In practice these awkward degeneracies almost never occur.

It is most likely, that these degeneracies are floating point based and have nothing to do with the classic description of the algorithm. As [16] states, the problem requires rational arithmetic to be solved. A quick floating point implementation should be avoided. When one is capable of using rational arithmetic, the implementation should be doable, when guided by [10] and [16].

During the span of this project, using CGAL or Computational Geometry Algorithms Library has been reserved as a back-up option, if implementing the minimal spanning disk/sphere algorithms would fail.

Cutting the geometry

Let us now give an explicit description on how the bounding-volume-hierarchy is partitioned. We start by fitting a minimal axis-aligned bounding rectangle (or bounding-box) around our model. Then we take the broadest dimension, and cut the model into two equally-sized parts. We create bounding boxes around the newly created parts and re-fit them such that they are minimal again. For our new two models, this process will go on recursively.

Because we are using axis-aligned bounding-boxes/bounding-rectangles, we are using an R-tree (as described in Section 3.4).

The partitioning of the geometry has to stop. It is not possible to simply give a upper bound on the number of polygons that may be present within a node in order to terminate the partitioning. In fact, it is possible to show, that with certain constructs, the number of polygons does not decrease. Sadly, one has to resort to more rigorous methods in order to stop the tree from exploding. One stopping criterion is that the maximum tree depth may not exceed 17 levels. Another stopping criterion is that when a leaf has a reached a certain size and the number of polygons is 4 or less, the partitioning must also stop.

The size criterion seems rather artificial. In practice, the size criterion forces the partitioning of polygons that are big. Whenever a section with a one big polygon ($\mathcal{O}(1)$) collides with multiple sections with small polygons ($\mathcal{O}(n)$) , we will still get a time complexity of $\mathcal{O}(n)$, which is still not really acceptable. By partitioning the big polygon into smaller parts, the bottleneck involving these

kind of collisions is reduced. Actually what will happen in the ideal situation is that $\mathcal{O}(n)$ is reduced to $\mathcal{O}(1)$.

A downside of the size criterion is that there is an additional parameter that influences the performance of the collision detection system.

Now that the geometry has been cut, for each node, a minimal spanning disk or sphere is computed. The computation of such an object takes $\mathcal{O}(n)$ expected time, where n is the number of vertices of which the geometry is composed. Each node represents the geometry of a model. The tree is built top-down. I.e. initially, the minimum spanning disk/sphere is computed around the full geometry. Then the geometry is cut into two parts, and the two minimal-spanning disks/spheres are computed for each part. Thus for each node, the minimal spanning disk/sphere is computed.

That each node represents the geometry of a model, does not mean that at each node geometry is stored. The geometry is stored only once, at the leaves (cut into pieces).

Quality of hierarchy construction

It is difficult to state much about storage requirements, and about the time it takes for the hierarchy to be constructed. It might be that these are asymptotically different for 2D and 3D. However, if we assume that the complexity of the geometry that is stored will only increase by a constant after chopping it up, then it is possible to make nice claims about the time complexity of the construction of the hierarchy.

If the latter assumption is true (it is just a guess actually), then because the depth of the tree is bounded and because the expected time of the minimal spanning sphere construction is $\mathcal{O}(n)$, this would imply that the (bounded)-tree can be built in linear expected time.

It should be noted that the above is quite deceptive. That the tree height is bounded is actually a weakness of the system. This weakness might cause high detail models having a highly variable level of detail to preform poorly. Yet this disadvantage is used to show that the asymptotic construction time is quite acceptable, while in practice the construction of the bounding volume hierarchy is slow, due to the usage of robust but inefficient ways of implementation.

Constructing two hierarchies in 3D for two 100K triangular models takes about a minute. In 2D, when supplying a set of random edges (instead of a set with decent topology), the construction time seems to explode. In this case, it seems to be that our uncertain assumption about the complexity of the geometry within the leaves is false.

It should be noted that in principle it is possible to pre-process the hierarchy and store it on disk. If the hierarchy would be stored in memory in an efficient manner, this might actually improve the performance of the continuous collision detection algorithm that reads from the hierarchy. Currently the way in which the hierarchy is stored in memory seems to be rather poor, i.e. each node of the tree has a memory address defined by a new memory allocation. The alternative is to use a single memory allocation to store a range of nodes.

5.4.2 An approximate sphere/sphere test

The proposed collision detection system relies heavily on approximate sphere/-sphere tests. The preliminary 2D version of the system, uses approximate disk/disk tests, but these rely on the usage of Taylor models. The eventual sphere/sphere tests that were used rely on the same function inclusion method that was used with the Taylor-Lagrange based root finder. There is no real good reason for the two approaches to be different, apart from the 3D approach not being propagated into the 2D project. The 3D approach involving second order Taylor-Lagrange-based models seems to be prettier than the 2D Taylor-model based approach.

It should be noted that the terminology of inclusion functions is rather confusing. Taylor models have been used in literature, and therefore the usage of the name is clear. Taylor-Lagrange-based inclusion functions rely on both Taylor's theorem and the Lagrange remainder. However, it must be noted that the Taylor models used within the 2D project also rely on the Lagrange remainder in order to bound the thickness of the models. Thus the naming is rather artificial. The original intent of Taylor models is that they are composed out of analytical functions. Within the 2D context of this project, the Taylor models are developed in a different way (similar to the way "Taylor-Lagrange models" are developed).

2D linear Taylor models

Suppose that we have a point, undergoing some kind of motion. We can construct a model that is valid along a certain time interval that bounds this point. This point trajectory can be seen as a parametrized vector function i.e. $\mathbf{p} : \mathbb{R} \mapsto \mathbb{R}^2$. For each of the two dimensions we can imagine a Cartesian system: $p_i : \mathbb{R} \mapsto \mathbb{R}$, $i \in \{1, 2\}$. Within this Cartesian system, we will see some function plot. This function plot can be very complicated, i.e. it may have oscillations. A linear Taylor model is simply a slab that is drawn around the Cartesian function plot. (A slab is simply a line-equation that is vertically fattened.) How do we draw this slab around the Cartesian plot? Since we define our Taylor model over an interval, we sample the function at the center of the interval. We sample the derivative of the function on the center of the interval, and we sample the second derivative over the entire interval (using interval arithmetic).

Next we draw a line tangential to the curve, trough the center of that curve. Thanks to the Lagrange remainder (see Appendix F), the error between the linearisation and the actual curve is made explicit. By using the bounds on the second derivative that we have obtained, it is possible to construct a lower bound and an upper bound on the error. These bounds are used to determine the vertical thickness of the slab.

This is done for each dimension of our Taylor model. If two 1D, first-order Tayor models bound the curve in the $x-$ and in the $y-$ dimension, than we have a single 2D-Taylor model bounding our 2D curve. Note that these models are computed in world-space.

In Figure 13, a snapshot of a moving shape is taken. The point indicated with a red cross has a trajectory and the first derivative of this trajectory is indicated with the blue line. The blue line only indicates the direction of the first derivative since it is cut by the bounds of the picture. The beam-like shape

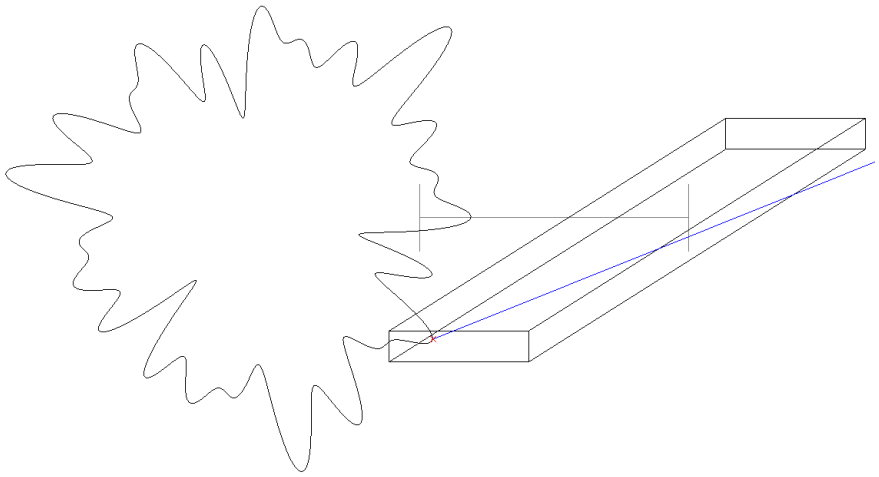

Figure 13: The beam-like shape represents a linear 2D Taylor model bounding the trajectory of the point that is indicated with a cross.

bounds the trajectory of the point on the shape associated with the red cross for approximately 0.5 seconds. If the time domain would be decreased, 0.25 seconds for instance, then the tightness of the bound would be roughly four times better. This is due to the square sign that is present within the error (thickness) of the linear Taylor models.

The description above is a bit incomplete. For instance, no hint is given on how to compute the intersection of two 2D-linear-Taylor-models. The reason why this section ends here is because the approach described here is not used anyway.

The alternative (proposed) method for approximate sphere/sphere intersection

Instead of constructing an inclusion function around a point in world space, this method is similar to a continuous feature test. Suppose that we have a point in the local coordinate system of an object that represents the origin of a sphere. If we are able to sample the 0^{th} the first and the second time-derivative of the origin, both at a certain t or a given time span, then this information can be put yo good use.

Given a sphere, and a trajectory of this sphere over a given time-domain, we construct a data-structure called "SphereOverDomain" that stores a number of things:

1. The radius of the sphere
2. The interval-origin of the sphere over the entire time domain
3. The interval-velocity of the sphere over the entire time domain

4. The interval-acceleration of the sphere over the entire time domain
5. The origin of the sphere at the beginning of the time domain
6. The velocity of the sphere origin at the beginning of the time domain
7. An axis-aligned bounding box bounding the swept volume of the sphere

Indeed, a lot of information is stored. Note that in order to obtain 4 one first needs to compute 3 and 2. Also note that 7 is determined by 2 and 1.

The information above is pre-computed and stored, because it is used multiple times. It may be required to make a single sphere collide with a few other spheres. If the information above would have to be computed twice for each sphere/sphere combination, then this would cause an excessive amount of computations. Note that by storing this information, we do not get an asymptotic improvement in performance. Finding all collisions between n spheres still takes $\mathcal{O}(n^2)$ time.

Suppose that we have two "SphereOverDomain" structures for two spheres, and suppose that we want to know the time of intersection between these two spheres. Similar to the example in section 3.4, we can do a simple test to avoid having to compute something difficult. Both bounding boxes of the swept volumes of the spheres are located in world space. By performing a simple bounding box intersection test, it is already possible to know that the spheres will not intersect within the time domain.

Suppose that the boxes do intersect, then something similar happens as in Section 5.2.1. Instead of using two lines or a point and a plane as primitives, it is also possible to use two spheres in a continuous test setting. The difference is that instead of applying a root finder as is done in Section 5.2.1, only a single iteration is performed. For this single iteration, it is possible to use the precomputed values stored in our "SphereOverDomain" structure. In Appendix B.2.2, a function is derived of which the roots describe the touching of the surfaces of two spheres. Such a function can be composed using two "SphereOverDomain" structures (requiring little additional effort). When the function itself, the first time derivative and the second time derivative over the time domain is computed, a second order Taylor-Lagrange based inclusion function is constructed (like is done in 5.2.1). We thus get two parabolas that bound the function.

Instead of computing an approximate lower bound for the TOC, also an upper bound is computed for the time after which the spheres will remain disjoint. Because the inclusion function is second order, this makes things a bit more complicated (i.e. the spheres may intersect, be disjoint, and intersect again). By using a few case distinctions, it is possible to account for all possibilities. Instead of returning a TOC, the sphere/sphere query returns a possibly empty time interval in which intersection may occur. Note that this result is an abstraction of the actual result.

When comparing the linear Taylor model approach with "SphereOverDomain" structure, the performance of both methods is the same (asymptotically). Because the linear Taylor models are developed in the middle of the time domain, the worst case bounding quality is four times better (roughly stated). However the bounding quality of the sphere over domain structure is perfect when considering the begin of the search domain. Whenever the sphere over domain structure is a "PointOverDomain" stucture, i.e. the radius of the sphere

is 0, and whenever t gets closer to the beginning of the time domain, the approximation gets better and better until it is perfect.

5.4.3 Traversing the hierarchy

Now that we have defined the bounding volume hierarchy, and because we have a broad-phase continuous collision detection primitive i.e. an approximate continuous sphere/sphere test, it is possible to traverse two bounding volume hierarchies simultaneously in order to perform continuous collision detection in an efficient way.

Let's first start with the beginning of the traversal. We have two trees and each tree has a base coordinate system and a linear translational and linear rotational motion associated with it. Thus we have two rotating and translating trees. Both trees have a root node. Each (root-) node has a bounding sphere. The origin of this bounding sphere is usually not the center of the object. First we apply an approximate sphere/sphere test as discussed in section 5.4.2. There are basically two possible outcomes. Either the approximate sphere/sphere test indicates non-intersection. In that case, we are done. The other option is that the approximate sphere/sphere test returns a time domain of possible intersection. In this case, we have to search for collisions within the time domain that was returned. The tree actually needs to be traversed.

Because the traversal of the two trees is a recursive process, this process is best described with pseudo-code:

Algorithm *Contact*(`aNode`, `bNode`, `aMotion`, `bMotion`, d)
1. **if** `aNode` is an empty leaf or `bNode` is an empty leaf
2. **then return nil**
3. `epoch` $\leftarrow d_{\text{lower}}$
4. $d \leftarrow [0, d_{\text{span}}]$
5.
6. `aMotion`.*Integrate*(`epoch`)
7. `bMotion`.*Integrate*(`epoch`)
8.
9. `nodePairList` $\leftarrow$ []
10. `result` $\leftarrow$ **nil**
11.
12. **if** `aNode` is a leaf and `bNode` is a leaf
13. **then** Preform raw collision detection on both leaves within domain d on `aNode` and `bNode` using `aMotion` and `bMotion`.
14. `result` $\leftarrow$ the TOC of the raw collision detection process (can be **nil**).
15. **else if** both `aNode` and `bNode` are not leaves
16. **then** Both nodes have two children. I.e. two spheres.
17. For each node, construct two "SphereOverDomain" structures.
18. I.e. one structure for the upper child and one structure for the lower child.
19. There are four "SphereOverDomain" structures in total.
20. Each child of node `aNode` can collide with each child of `bNode`.
21. Thus there are four combinations that can collide.
22. Add all four combinations to `nodePairList`.

```
23. else if aNode is a leaf and bNode is not a leaf
24.    then Now we have an asymmetric situation.
25.         Construct a "SphereOverDomain" for aNode
26.         Construct two "SphereOverDomain" structures, one for the lower
            child of bNode and one for the upper child of bNode.
27.         There are thus three sphere over domain structures.
28.         Also, there are two possible interactions.
29.         Add the two possible interactions to the node pair list.
30. else if aNode is not a leaf and bNode is a leaf
31.    then Construct a "SphereOverDomain" structure for bNode.
32.         Construct two "SphereOverDomain" structures for the two chil-
            dren of aNode.
33.         Add all two interactions to nodePairList.
34.
35. For each element within nodePairList compute the domain of intersection
    (may be empty) using the approximate sphere/sphere test. And store this
    domain of intersection within the same element.
36. Prune the entries in nodePairList that have an empty domain of intersec-
    tion.
37. These entries can be considered non-colliding.
38.
39. if Size(nodePairList) ≠ 0
40.    then Sort nodePairList on approximate TOC ascendingly.
41.         The approximate TOC is the lower bound of the domain of inter-
            section.
42.
43.         maxTOC ← d_upper
44.
45.         for i ← 1 to Size(nodePairList)
46.             do if result ≠ nil
47.                   then maxTOC ← min{maxTOC, result}
48.
49.                minTOC ← the approximate TOC (lower-bound) of nodePairList[i]
50.                timeOfNonContact ← the time after which contact cannot
                   be regained within d considering nodePairList[i]
51.                maxTOCforThisNodePair ← min{maxTOC, timeOfNonContact}
52.                if minTOC ≤ maxTOCforThisNodePair
53.                   then
54.                        a ← the node of object A associated with nodePairList[i]
55.                        b ← the node of object B associated with nodePairList[i]
56.                        newD ← [minTOC, maxTOCforThisNodePair]
57.                        subResult ← Contact(a, b, aMotion, bMotion, newD)
58.                        if result = nil
59.                           then result ← subResult
60.                           else if subResult ≠ nil
61.                                   then result ← min{result, subResult}
62.
63. if result = nil
64.    then return nil
65. return result + epoch
```

Immediately it should be noted that a motion object, is actually a state of the physical object. A state that is allowed to be advanced by integrating it. The goal of the function above is to search for collisions within time-domain d, between `aNode` and `bNode`, having `aMotion` and `bMotion` respectively.

Note that the "current" time is basically advanced to `epoch`. The advantage of advancing the time is that the upper bound on the time domain gets smaller. This seems to have a positive influence on the precision of the root-finder that is used to perform the raw-collision detection. Leaving out the epoch translation would be an option since the above is only pseudo-code, but a choice has been made to approximate the actual code as close as possible. The cost of integrating `aMotion` and `bMotion` a lot of times is negligible.

It should also be noted that the algorithm above is only about TOC queries. TOC queries are the most important part of continuous collision detection. However, with a slight modification, i.e. interpreting `result` as a "ContactResult" object instead as an element of $\mathbb{R} \cup \{\textbf{nil}\}$, it is possible to also query information such as a single contact point and associated contact normal.

The response time of the above is indecent

When we want to collide two objects, and basically call the function preceding *Contact* (i.e. the function testing the root-node spheres of both trees, a function that has not been explicitly declared), then the user is likely to get a extremely poor performance. The latter has to do with the fact that the time domain d that is supplied is simply too large for the approximate sphere-sphere tests. If the span of d is too large, then the parabolas bounding the distance function are not tight enough. This may result in false approximate collisions being declared by the approximate sphere/sphere test, or it may be the case that the approximate TOC that was found is way too low.

If line 14 is replaced by `result` $\leftarrow 0$, then we do not concern ourselves with raw collision detection any more. In fact, all that is remaining is a bounding-volume-hierarchy that is traversed, giving us a lower-bound on the actual TOC. If a big domain is supplied to the root collision detection procedure and if we are only concerned with the bounding volume hierarchy, then we can simply observe the result of the approximate collision detection query.

What is immediately clear is that the objects have advanced only a bit, and are totally not touching each-other. The continuous sphere/sphere tests are conservative, thus the advancement is conservative (not the be confused with conservative advancement). Because the time domain is too large, this will cause big values for the second time-derivative interval of the Taylor-Lagrange based inclusion functions. By making the time-domain smaller, the Taylor-Lagrange based inclusion functions will produce a better bound.

This problem applies to both Taylor models and the Taylor-Lagrange based inclusion functions. In the 2D project, this has been solved by simply picking a constant c, and by chopping up the domain into c parts. Thereby executing the recursive process many times, for a single collision. Some effort has been undertaken to use multiple iterations for an approximate sphere/sphere test, but this did not seem to be a good solution, due to the fact that by using a single iteration, it is possible to re-use the "SphereOverDomain" structure multiple times. Additionally it should be mentioned that the re-use of this structure was only done in 3D. This can be seen as a clear reason why the 2D project is

non-optimal.

One might expect that executing the root collision detection procedure n times, would multiply the time spent by $O(n)$. This is definitely not the case. Because the time domain is smaller for each call, this drastically improves the bounding quality of the approximate sphere/sphere tests. Thus it is more likely that children of nodes located close to the root in the hierarchy, indicate non-intersection, excluding lots of potential collisions.

The next question is: how big should n be? This question is rather hard, because it is not only dependent on the time domain, it is also dependent on other factors such as the translational and rotational speed of both objects. Because we can consider the scene to be kind of dimensionless, it is also preferable to take the relative complexity of the objects with respect to their size into account.

Let us consider a simple example in order to explain this. Suppose that we have two triangulated spheres each consisting out of m triangles. Suppose that the spheres represent the earth and the moon. All units are based on meters. Now suppose that we have a model of some miniature stellar set-up, that is precisely identical, using the same triangular models but then scaled down. Of course we want the miniature set-up to use the same n as the earth and the moon. However, one cannot achieve this by simply considering the relative velocity between both objects. It is unwanted that these two set-ups that are more or less identical would have differing performance. In the current system, the performance does differ, but this is due to the size stopping criterion of the bounding volume hierarchy.

Attempts have been made to compensate for these phenomena. It must be noted that this has been the subject of experimentation. It can be argued that the function for determining n can be more efficient than it is now.

Algorithm *ComputeN*(`aNode`, `bNode`, `aMotion`, `bMotion`, d, `constant`)

1. `secondDerivativeMagnitude` $\leftarrow 0$
2. `secondDerivativeMagnitude` $\leftarrow \max\{$`secondDerivativeMagnitude`,
3. `aNode`.*getMaxRadius*$() \cdot \|$`aMotion`.$\omega\|_2^2\}$
4. `secondDerivativeMagnitude` $\leftarrow \max\{$`secondDerivativeMagnitude`,
5. `bNode`.*getMaxRadius*$() \cdot \|$`bMotion`.$\omega\|_2^2\}$
6. `wostCaseVelocityA` $\leftarrow 0$
7. `wostCaseVelocityA` $\leftarrow$ `wostCaseVelocityA` $+ \|$`aMotion`.$\nu\|_2$
8. `wostCaseVelocityA` $\leftarrow$ `wostCaseVelocityA`$+$`aNode`.*getMaxRadius*$()\cdot\|$`aMotion`.$\omega\|_2$
9. `wostCaseVelocityB` $\leftarrow 0$
10. `wostCaseVelocityB` $\leftarrow$ `wostCaseVelocityB` $+ \|$`bMotion`.$\nu\|_2$
11. `wostCaseVelocityB` $\leftarrow$ `wostCaseVelocityB`$+$`bNode`.*getMaxRadius*$()\cdot\|$`bMotion`.$\omega\|_2$
12. `worstCaseVelocity` $\leftarrow \max\{$`worstCaseVelocityA`, `wostCaseVelocityB`$\}$
13. `correctionScalar` $\leftarrow \max\{$`secondDerivativeMagnitude`$\cdot 0.5$, `worstCaseVelocity`$\}$
14. `minimumObjectRadius` $\leftarrow \min\{$`aNode`.*getMaxRadius*$(),$ `bNode`.*getMaxRadius*$()\}$
15. `minimumObjectRadius` $\leftarrow \max\{$`minimumObjectRadius`, $\epsilon_m\}$
16. **return** $\max\left\{\left\lceil \frac{d_{\text{span}} \cdot \texttt{correctionScalar} \cdot \texttt{constant}}{\texttt{minimumObjectRadius}} \right\rceil, 1\right\}$

Where *getMaxRadius* denotes the maximum radius of an object with respect to its center of gravity (this is not the radius of the bounding sphere). Line 15 is just there to prevent a division by 0. The variable `constant` is an arbitrary constant that needs to be determined experimentally. It should be noted that

the function above has been determined by the application of trial and error. There is some rationale behind the function, although it is not fully logical.

5.4.4 Capoeira mode

One may have noticed that in order to test the effectiveness of the bounding volume hierarchy, the raw-collision detection was omitted. This eventually yielded a system that is very effective at computing a lower-bound for the TOC. Without considering raw collision detection, this system already produced results that were in some cases comparable in accuracy to the system that was benchmarked against (Controlled Conservative Advancement). Because the system does not consider raw-collision detection, which is definitely the bottleneck, the system is faster at the expense of accuracy. This approximate method for continuous collision detection may be useful in certain cases. Though there are cases in which the method totally fails in accuracy (as described in section 7.4.1). This approximate method is designated with the name Capoeira mode, named after the Latin-American martial-art in which one strives not to hit his or her opponent.

Let us now list all the systems:

- Differentiate Twice Method (D2M)
  This is the method presented in this thesis. You have to analytically differentiate an arbitrary point on the object twice (with respect to time), if you want a customized motion.
- Differentiate Twice Method, Capoeira (D2M Cappoeira)
  This is the quick method discussed above.
- Controlled Conservative Advancement (C2A)
  D2M is benchmarked against this system.

In Figure 14, one can see nodes of the tree at the levels 0 (top-left), 4 , 8 and 12. Note that the all the geometry of the (Stanford-) bunny is contained within the nodes.

It is possible to pose the question whether Capoeira mode is a fully continuous method. Capoeira mode does not miss collisions, though declaring false positives is inherent to the method. With other methods like D2M and C2A, the false positives of the system can be neglected as the distance epsilon of the system converges to 0. Thus false positives of D2M and C2A are caused by numerical inaccuracies, while false-positives of D2M Capoeira are caused by both numerical inaccuracies and also because the model has been deformed (enlarged). Within Section 7, D2M Capoeira is treated as if it is fully continuous. The inherent deformations are accounted for as inaccuracy of the system.

For Capoeira mode, the function *ComputeN* is called in order to determine the number of subdivisions. The parameter `constant` has been experimentally minimized, such that the accuracy is within acceptable levels. In fact it is chosen such that the accuracy does not improve significantly if it was raised. This approach has shown to be very effective. It is that effective, that even the precise collision detection system is based on Capoeira mode. The latter statement requires additional explanation.

By first advancing the objects with Capoeira mode, it is possible to make an additional optimization to the system. When the objects have been advanced,

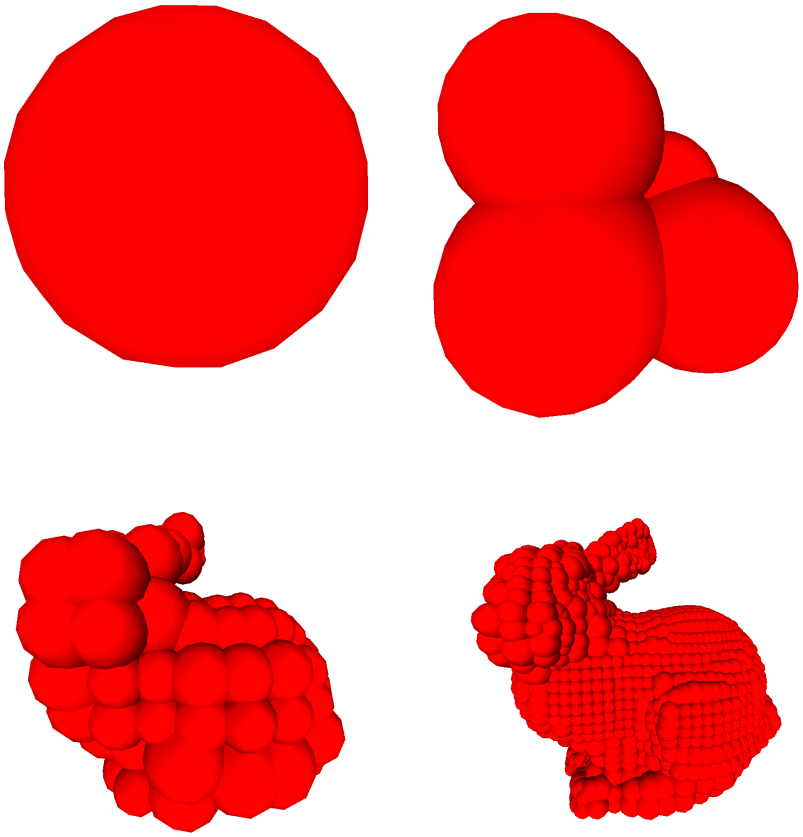

Figure 14: Visualizing the D2M tree at different levels.

the remaining tiny part is handled by a continuous collision detection query, that is capable of using raw collision detection. This remaining query is associated with a much smaller domain. This query also uses the *ComputeN* function, but with a different `constant` parameter. Actually, the `constant` is 512 times bigger than the `constant` parameter used in Capoeira mode. Decreasing the size of the time-domain for near-contact, seems more computationally efficient. Decreasing the time domain, may help prune some expensive raw collision detection tests. It may also shorten the time domain of the root finder, causing faster convergence.

There is one other reason why shortening the time span may improve performance. That is because increasing the number of time-domains, causes the recursive call to synchronize at fixed moments in time. To give a more concrete example, suppose that at the two root nodes of A and B, all four children, from two pairs of interactions (actually there can be four pairs of interaction, but suppose these are pruned). What will happen is that the problem is split into two parts. It may be the case that the part that is executed first returns a TOC of 1.1, while the second part returns a TOC of 1.0, thereby discarding the 1.1 TOC and perhaps some additional effort to find a TOC that is between the approximate TOC and 1.1. To state things briefly, the raw TOCs that are found

are not always found in the same order in which the algorithm is executed.

The approximate TOCs that are determined by the sphere/sphere tests, cause the order of execution of the algorithm. Within literature [18], it is described that the order of execution of such an algorithm can be determined by the Euclidean distance between the nodes at a certain moment in time. This approach has not been investigated within the context of this project. In theory, using both the Euclidean distance and the approximate time of contact to determine the order of traversal may lead to a better system.

Whenever two leaves collide, there is an additional construct that guards against brute-force collision detection.

5.4.5 Brute-force optimization of raw-collision detection

The raw-collision detection procedure has a time complexity of $\Theta(A_f \cdot B_f)$ (assuming that a feature test takes constant time). However, the time it takes can be written as a quadratic polynomial. It is possible to make the linear term bigger and the quadratic term smaller. This is done by caching the "SphereOverDomain" structures for each feature within the leaf of object A and for each feature within the leaf of object B. In practice, two vertex lists are constructed, two polygon lists, and two edge lists, each element of each list containing a "SphereOverDomain" structure that bounds the feature inside. We thus avoid having to recompute these structures for each feature/feature interaction that occurs. A continuous intersection test between two "SphereOverDomain" structures is relatively cheap. If there is no intersection, a really expensive raw feature test can be avoided.

5.4.6 Reasons why the system is non-optimal

There are various reasons why the system can be improved upon. Some of them are listed below:

1. There exist edges that are created by cutting the geometry. Some of these edges are pruned after the construction of the hierarchy, but not all of them. It is even allowed to prune edges of two adjacent polygons of which the angle in-between is insignificant. By pruning all non-significant edges, the system can be optimized.

2. The geometry is cut, in order to obtain a reasonably tight bounding volume hierarchy. For example, a single triangle can be cut in many different parts each fitted into a bounding sphere of a leaf node. Instead of storing the geometry within the leaf nodes, it is possible to store the uncut geometry associated with the hierarchy. Within each leaf node it would be possible to make references to the geometry that falls inside. The geometry that has been cut can be discarded saving lots of storage space. The hierarchy (including minimized bounding spheres) remains.

3. In-efficient in-memory representation of the bounding volume hierarchy. The bounding volume hierarchy was constructed with the focus on correctness and asymptotic construction performance. In practice the construction is rather slow. This does not matter that much, since it only needs to be done once for a rigid body. Then again, lots of C++ standard

template library components have been used like red-black trees to achieve high levels of abstraction. However using these components comes at the cost of lots of memory indirections. Reimplementing the bounding volume hierarchy, considering item 2 and the cost of memory indirections could decrease the construction time, the storage requirements and ultimately increase the actual performance of the bounding volume hierarchy.

6 Constructing a Rigid Body Simulation

Now that a continuous collision detection system has been constructed, in order to be able to make a simulation, one has to deal with three things.

1. Unconstrained Rigid Body dynamics
2. Resolving a collision with impulse
3. Sustaining a collision with normal forces

Instead of constructing a rigid body simulator that is capable of simulating n rigid bodies, let us limit ourselves to two rigid bodies. Unconstrained dynamics are debated in [3], as well as resolving collisions with impulse and sustaining a collision with normal forces. However, 3 is solved using quadratic programming. Because quadratic solvers can be seen as kind of specialized equipment, it is preferable to use a linear (inequality) solver to obtain a solution. This is done in [7]. In [7], Erin Catto uses a projective Gauss-Seidel solver in order to solve 3. The method is generic in the sense that it is capable of solving the contact forces between n rigid bodies. Any number of rigid bodies can be in simultaneous contact, when solving the normal forces between these bodies, one has to consider all the bodies.

That three bodies collide at the exact same moment in time is rather exotic. Usually, when two objects collide, there is a single contact point at the TOC, which can be used to resolve the collision with impulse. However, this is not always the case. One can think of two cubes that are frontally colliding.

> When gravity is present, it is not that uncommon, that three objects share resting contact. This is not treated within this thesis.

If one chooses to use one contact point, the cube will bounce off, but not in the way that is preferred. Instead of manipulating the linear velocity of the cube, the angular velocity is changed. This change in angular velocity, will almost immediately cause a second collision event. This collision event will compensate for the change in angular velocity. Instead of getting one collision, we get a sequence of immediately following collisions. This sequence of immediately following collisions puts a strain on the collision detection system. Because of this, it is preferred to determine all contact points between the two bodies, and solve the impulse for all contact points simultaneously.

The problem of resolving a collision with impulses considering multiple contact points is treated in [4]. Here they describe the rationale (the physics) of how such a collision is resolved. They also give an algorithm that describes how the impulses are computed. However, this description is rather vague, therefore instead of using their algorithm, a custom algorithm is derived in section 6.2.

The algorithm in this section, is based on the physics described in [4], however instead of using the iterative algorithm that they suggest, a projected Gauss-Seidel solver ($\mathbf{Ax} = b$ s.t. $x_i \geq 0$ for $i \in \{1 \dots n\}$) is used.

6.1 Determining contact points

When we have the time of contact, we need contact points. Actually, we need contact-points and associated normals. The normals can usually be inferred, however there are some complexities with edge/edge contacts.

6.1.1 Determining additional contact points in Capoeira mode

In Capoeira mode, there are two leaves that are close to each other. These leaves are not necessarily spherical as one might think. In fact, they are the intersection of many spheres. We do have an origin of the leaves that are hitting each-other. One can compute the distance in between and multiply it by a constant factor let us say $\frac{3}{2}$. Next a static proximity query is done to find all the nodes that are nearby. The custom BVH that was constructed can also be used for proximity queries on nodes. The construct is not entirely proper, but good results are obtained.

"Do nodes overlap when they are enlarged?".

Additionally, it must be noted, that for determining the normals of Capoeira mode, a very primitive method was used. (I.e. it is not the method of choice.)

6.1.2 Determining additional contact points considering the full geometry

With the full geometry, things are less complicated. It is possible to query the precise time of contact (not caring about minute penetrations). The next thing to do is take a tiny step back, such that there is a minimum distance between the objects. This can be done, because we already have one contact point and associated contact normal. Taking a step back can be done by decrementing the TOC the right way. Next a proximity query is performed. Both models are enlarged. If the leaves overlap, the geometry inside is tested for approximate intersection.

6.2 Solving the impulse for a number of contact points between two bodies

It must be noted that this section leads to a linear system of which the solution is iteratively clamped to the non-negative domain. The construct does not give the right output for all input. This can be, because the solution is non-unique or because the system does not obey the criteria for the Gauss-Seidel solver that is used.

It seems that the approach described in [4], is better. The construct described below guarantees nothing. Whereas [4] is written by experts. Their construct (a numerical algorithm) is difficult to understand.

The algorithm below is the simultaneous application of Newton's impact law on multiple contact points. The solution, λ_i, is a vector that has non-negative elements. Each element corresponds with a contact point and associated contact normal. The construct is similar to the construct of [3]. The only difference is that in this case multiple contact points are handled simultaneously.

We have two rigid bodies at the instance of collision. There are n contact points at the approximate instance of collision. We only have to consider those contact points, which are actually tending towards penetration, meaning that their relative velocity is contrary to the normal direction associated with the contact point.

To be precise; let the input be defined by the following values: $\mathbf{p}_A^-$, $\mathbf{L}_A^-$, $\mathbf{p}_B^-$, $\mathbf{L}_B^-$, m_A, I_A,m_B,I_B,$\mathbf{x}_A$,$\mathbf{x}_B$. Where $\mathbf{p}_O^-$ and $\mathbf{L}_O^-$ respectively denote the linear

and angular momentum of object O. I_O and m_O respectively denote the inertia tensor and the mass of object O. Whereas $\mathbf{x}_O$ denotes the centroid of object O. Note that the orientation of the object is left out. This is because we assume that the inertia tensors I_A and I_B are defined in world coordinates.

Let the contact points be defined as follows; let $\mathbf{c}_i \in \mathbb{R}^3$ be a contact point, and let $\hat{\mathbf{n}}_i \in \mathbb{R}^3$ be its associated normal. Let $\mathbf{r}_{O,i}$ be the arm of contact point $\mathbf{c}_i$ with respect to object O. Thus $\mathbf{r}_{O,i} = \mathbf{c}_i - \mathbf{x}_O$.

Let $\mathbf{v}_O^-$ be the velocity of the centroid of object O before the collision has been resolved, i.e. let $\mathbf{v}_O^-$ be $\frac{1}{m_O}\mathbf{p}_O^-$. Let ω_O^- be the angular velocity before the collision has been resolved, i.e. let ω_O^- be $I_O^{-1}\mathbf{L}_O^-$.

We can describe the velocity at a certain $\mathbf{c}_i$ on a certain object. Since we have two objects, we thus have two velocities at $\mathbf{c}_i$. The velocity of point $\mathbf{c}_i$ can be seen as the rate of change of $\mathbf{c}_i$ with respect to time and with respect to a certain object before the collision has been resolved.

We can thus define $\frac{d\mathbf{c}_{O,i}^-}{dt}$ as $\mathbf{v}_O^- + \omega_O^- \times \mathbf{r}_{O,i}$. Let the relative velocity at $\mathbf{c}_i$ before the collision be $\frac{d\mathbf{c}_{B,i}^-}{dt} - \frac{d\mathbf{c}_{A,i}^-}{dt}$. A criterion for any contact point $\mathbf{c}_i$ with associated normal $\hat{\mathbf{n}}_i$ is that $(\frac{d\mathbf{c}_{B,i}^-}{dt} - \frac{d\mathbf{c}_{A,i}^-}{dt}) \cdot \hat{\mathbf{n}}_i \leq 0$, thus to have a tendency to cause penetration.

Now that we have described the input, let us describe the output. The output can be seen in multiple ways. Meaning that the output can have an extended form (it can be seen as a collection of non-negative scalars i.e. λ_i, or it can be seen as the difference in momentum (both angular and linear) of both objects.

We want the objects to bounce off each other in an approximately correct manner. This can be achieved by interpreting Newton's impact law in a non-conservative manner (as is done in [4]). For each contact point, there is an initial relative velocity i.e. $(\frac{d\mathbf{c}_{B,i}^-}{dt} - \frac{d\mathbf{c}_{A,i}^-}{dt}) \cdot \hat{\mathbf{n}}_i$. Newton's impact law is a statement about the change in relative velocity: $(\frac{d\mathbf{c}_{B,i}^+}{dt} - \frac{d\mathbf{c}_{A,i}^+}{dt}) \cdot \hat{\mathbf{n}}_i = -e(\frac{d\mathbf{c}_{B,i}^-}{dt} - \frac{d\mathbf{c}_{A,i}^-}{dt}) \cdot \hat{\mathbf{n}}_i$, where e is the coefficient of restitution i.e. $e \in [0, 1]$. To give an example, $e = 0$ implies a plastic collision, whereas $e = 1$ implies an elastic collision.

In order for the moment of collision to be resolved, each contact point/normal pair must exert a symmetric impulse on both bodies. Meaning that an impulse applied on $\mathbf{c}_i$ in the direction of $\hat{\mathbf{n}}_i$ acts on object B. And this same impulse will act on object A in the direction of $-\hat{\mathbf{n}}_i$. Let the magnitude of these impulses be undetermined, and let us call these magnitudes λ_i. Because we have to solve a magnitude, and not a scalar, we have the additional constraint that all λ_i must be non-negative.

Let n be the number of contact points and let $f : \mathbb{R}^n \mapsto \mathbb{R}^{12}$ be a (linear) transformation that maps impulse magnitude to delta momentum. To be explicit $f(\lambda) = \delta\mathbf{s} = \begin{pmatrix} \delta\mathbf{p}_A \\ \delta\mathbf{L}_A \\ \delta\mathbf{p}_B \\ \delta\mathbf{L}_B \end{pmatrix}$.

Let us be explicit on the meaning of $\delta\mathbf{p}_O$ and $\delta\mathbf{L}_O$. Both are the difference in momentum ($\delta\mathbf{s}$) that is applied i.e. $\mathbf{p}_O^+ = \mathbf{p}_O^- + \delta\mathbf{p}_O$ and $\mathbf{L}_O^+ = \mathbf{L}_O^- + \delta\mathbf{L}_O$. Where $\mathbf{p}_O^+$ and $\mathbf{L}_O^+$ denote the linear and angular momentum after the collision has been resolved. Stated otherwise $\mathbf{s}^+ = \delta\mathbf{s} + \mathbf{s}^-$.

Let $g : \mathbb{R}^{12} \mapsto \mathbb{R}^{n3}$ be an affine transformation (i.e. having the shape $\mathbf{Ax}+\mathbf{b}$), that maps the $\delta\mathbf{s}$ of both objects to a vector of relative velocities, corresponding with all the contact points. Let h be a function that measures the relative velocities in the direction of the associated contact normal.

We can now describe the system by using the three functions above:

$$h(g(f(\lambda))) = \mathbf{u}. \tag{2}$$

Subject to $\lambda_i \geq 0$. Where $\mathbf{u}$ is a vector containing the resultant relative velocities in the direction of the associated normal. These relative velocities are pre-computed, by considering each contact point individually using Newton's impact law.

Let us first describe the linear transformation f. Let $\mathbf{A}_i$ be a row vector having 12 elements:

$$\mathbf{A}_i = (-\hat{\mathbf{n}}_i^T, -(\mathbf{r}_{A,i} \times \hat{\mathbf{n}}_i)^T, \hat{\mathbf{n}}^T, (\mathbf{r}_{B,i} \times \hat{\mathbf{n}}_i)^T) \tag{3}$$

Let the transformation $f(\lambda)$ be equal to $\mathbf{A}^T\lambda$. The latter is true because of the following equations:

$$\begin{aligned}
\delta\mathbf{p}_A &= -\sum_i \lambda_i \hat{\mathbf{n}}_i & (4)\\
\delta\mathbf{L}_A &= -\sum_i \mathbf{r}_{A,i} \times (\lambda_i \hat{\mathbf{n}}_i) & (5)\\
\delta\mathbf{p}_B &= \sum_i \lambda_i \hat{\mathbf{n}}_i & (6)\\
\delta\mathbf{L}_B &= \sum_i \mathbf{r}_{B,i} \times (\lambda_i \hat{\mathbf{n}}_i) & (7)
\end{aligned}$$

Before describing transformation g, let us first describe the equations that map the delta momentum to the relative velocities at the contact points. In order to do this, let us first describe how to compute the values for $\mathbf{v}_O^+$ and ω_O^+.

$$\begin{aligned}
\mathbf{v}_O^+ &= \mathbf{v}_O^- + \frac{1}{m_O}\delta\mathbf{p}_O & (8)\\
\omega_O^+ &= \omega_O^- + I_O^{-1}\delta\mathbf{L}_O & (9)
\end{aligned}$$

We can define the relative velocity as $\frac{d\mathbf{c}_{B,i}^+}{dt} - \frac{d\mathbf{c}_{A,i}^+}{dt}$ which equals:

$$\begin{aligned}
&(\mathbf{v}_B^+ + \omega_B^+ \times \mathbf{r}_{B,i}) - (\mathbf{v}_A^+ + \omega_A^+ \times \mathbf{r}_{A,i}) =\\
&\mathbf{v}_B^+ + \omega_B^+ \times \mathbf{r}_{B,i} - \mathbf{v}_A^+ - \omega_A^+ \times \mathbf{r}_{A,i} =\\
&\mathbf{v}_B^+ - \mathbf{r}_{B,i} \times \omega_B^+ - \mathbf{v}_A^+ + \mathbf{r}_{A,i} \times \omega_A^+ =\\
&\mathbf{v}_B^+ - \mathbf{r}_{B,i} \times I_B^{-1}\mathbf{L}_B^+ - \mathbf{v}_A^+ + \mathbf{r}_{A,i} \times I_A^{-1}\mathbf{L}_A^+ =\\
&\mathbf{v}_B^+ - \mathbf{r}_{B,i}^* I_B^{-1}\mathbf{L}_B^+ - \mathbf{v}_A^+ + \mathbf{r}_{A,i}^* I_A^{-1}\mathbf{L}_A^+ =\\
&\tfrac{1}{m_B}\mathbf{p}_B^+ - \mathbf{r}_{B,i}^* I_B^{-1}\mathbf{L}_B^+ - \tfrac{1}{m_A}\mathbf{p}_A^+ + \mathbf{r}_{A,i}^* I_A^{-1}\mathbf{L}_A^+
\end{aligned}$$

Where $\mathbf{r}_{O,i}^*$ represents a matrix that has the same effect as the operation $\mathbf{r}_{O,i} \times \ldots$. It must thus be noted that $\mathbf{r}_{O,i}^* I_O^{-1}$ is a matrix. The resultant equation can be written in matrix form:

$$\frac{d\mathbf{c}_{B,i}^{+}}{dt} - \frac{d\mathbf{c}_{A,i}^{+}}{dt} = \left(-\frac{1}{m_A}\mathbf{E}_3, \mathbf{r}_{A,i}^{*} I_A^{-1}, \frac{1}{m_B}\mathbf{E}_3, -\mathbf{r}_{B,i}^{*} I_B^{-1} \right) \mathbf{s} = \mathbf{B}_i \mathbf{s}^{+}. \tag{10}$$

Where $\mathbf{E}_3$ is the 3×3 identity matrix. Note that $\mathbf{B}_i \in \mathbb{R}^{3\times 12}$. The full matrix $\mathbf{B}$ is thus an element of $\mathbb{R}^{3n\times 12}$. This can be rewritten into:

$$\mathbf{B}\mathbf{s}^{+} = \mathbf{B}(\mathbf{s}^{-} + \delta\mathbf{s}) = \mathbf{B}\delta\mathbf{s} + \mathbf{B}\mathbf{s}^{-} \tag{11}$$

Our function $g(\delta\mathbf{s})$ is thus defined as $\mathbf{B}\delta\mathbf{s} + \mathbf{B}\mathbf{s}^{-}$. Note that the term $\mathbf{B}\mathbf{s}^{-}$ can be pre-computed.

Now we have n relative velocity vectors. These velocity vectors still need to be translated into velocity scalars (in the direction of the associated normals). This is done with the mapping function h or its matrix variant $\mathbf{N}$.

Let $\mathbf{N} \in \mathbb{R}^{n\times 3n}$, and let it be defined as follows:

$$\mathbf{N} = \begin{pmatrix} \hat{\mathbf{n}}_1^T & \mathbf{0}^T & \mathbf{0}^T \\ \mathbf{0}^T & \dots & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{0}^T & \hat{\mathbf{n}}_n^T \end{pmatrix}. \tag{12}$$

Our initial equation now translates to:

$$\mathbf{N}(\mathbf{B}(\mathbf{A}^T\lambda) + \mathbf{B}\mathbf{s}^{-}) = \mathbf{u}. \tag{13}$$

After rewriting this equation, this yields:

$$\mathbf{N}\mathbf{B}\mathbf{A}^T\lambda = \mathbf{u} - \mathbf{N}\mathbf{B}\mathbf{s}^{-}. \tag{14}$$

The matrix $\mathbf{N}\mathbf{B}\mathbf{A}^T$ can be precomputed, as can the vector $\mathbf{u} - \mathbf{N}\mathbf{B}\mathbf{s}^{-}$. The strategy is to solve λ under the criterion $\lambda_i \geq 0$ by using a projected Gauss-Seidel solver. It is unclear whether the system will converge to a unique solution. This is sometimes not the case, because it may depend on the input. In [7] an example of a set of redundant constraints is given causing multiple solutions to be valid.

Although the system does not strictly fulfil the criteria that are required for the projected Gauss-Seidel solver to work properly, it is possible to obtain good results in practice. This may require a lot of tweaking though. Resting contact is not treated well. In fact resting contact is modelled as tiny bounces, causing the system to use too much processing power.

7 Results

7.1 Preliminary Setup

The initial results of this master project were as follows:

> Consider two disjoint rotating objects in the plane. These objects are fixed with their center of gravity to the horizontal axis. For each moment in time (i.e. discrete moments in time), these two objects are continuously pushed against each-other, in order to illustrate the continuous collision detection method. What remains is a scene of two rotating objects that are sticking to each-other (see Figure 15). This process was performed for two identical models of both 65536 edges and ran at approximately 200 collisions per second (on average). This benchmark was preformed on a single thread on an Intel® Core™ i7-2670QM CPU @ 2.20GHz × 8.

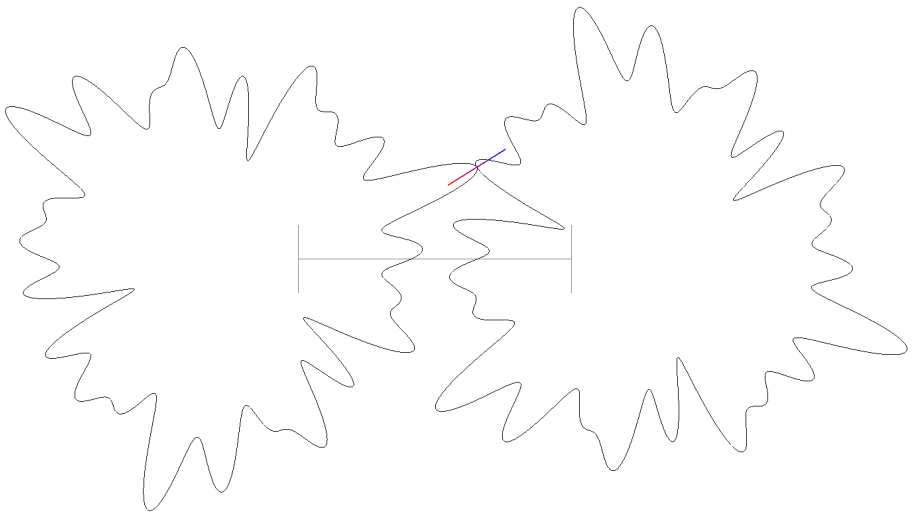

Figure 15: Two "curved" objects that are pushed together.

One might have noticed that the initial results were results in the plane, thus 2D and not 3D. It would be a reasonably safe guess that performing collision detection in 3D is computationally more expensive than performing it in 2D. It should also be noted that this test does not involve continuous rotations, although the visual result does imply that. However, the method implemented fully supports continuous rotational and translational motion.

It is very questionable whether the preliminary test set-up is good. Each frame, an entire time span of continuous collision detection is performed (in this case, a time span of one second). The test can be altered to make the results

look better or worse. Adding e.g. continuous rotations has a negative effect on the number of collisions per second.

Now, suppose that there is an animated scene that spans one second. And suppose that in this one second, there is just one collision. Then most likely, all of the non-colliding frame intervals will be very computationally efficient. The actual collision may be very expensive, but on average, the "benchmark" may still reach many thousands of frames per second.

This aspect of simulations has already been identified by many authors and this is partially solved by denoting the computation time per frame (usually in milliseconds). Little can be said about the preliminary results of this master project. More aspects of the preliminary set-up can be described, however since this has little purpose such a description is reserved for the eventual set-up. Not much effort has been undertaken to make elaborate benchmarks of the 2D setup.

7.2 Validation

Because it is very difficult to compare apples and oranges, the eventual result of this project needs to be compared with an existing (state of the art) system that will belong to the same class of systems as the system to be devised.

Controlled Conservative Advancement (C2A)

Controlled Conservative Advancement is a system that can preform continuous collision detection on polygon soups. In [20], the authors do not explicitly show that the method is capable of colliding two rigid bodies that are both under interpolated rigid motion. When they present their method, they always tend to keep one object fixed. This simplifies the problem up to a certain extent, however their continuous collision detection application programming interface seems to approximately give the right output when using two bodies that are under interpolated rigid motion. Also the source code of their product suggests that the method is intended to deal with two rotating and translating rigid bodies.

Both C2A and the proposed method have to be comparable, they have to solve exactly the same problem. Namely a TOC query between two rotating and translating objects.

Box2D

Before this validation can occur, the system needs to be completed. Within this project, this has been done in two phases, namely the 2D phase and the 3D phase. Initially it was the intent to verify the 2D result against a 2D continuous collision detection system called Box2D[4]. Because Box2D is an entire 2D-physics solution, and the 2D results of this project only incorporate collision detection, Box2D as a whole is incomparable to the 2D results of this project. The Box2D system seems well optimized, making it more difficult to separate its components. Another aspect is that Box2D is optimized for the collision of many simple objects, instead for the collision of complicated objects (which is

[4] `http://box2d.org`

a goal of this thesis). At least, a demo program of Box2D only shows simple box-like shapes. If the system was well capable of colliding complicated objects, then this would most likely have been demonstrated. Box2D is capable of colliding non-convex polygons however, the number of edges always seems to be low.

Concrete results have been achieved in a 3D setting. The system that has been devised, Differentiate Twice Method, is compared to Controlled Conservative Advancement.

The Controlled Conservative Advancement source code is available for educational, research and non-profit purposes. The system has been successfully tested under Microsoft Windows using Visual C++ 9.0. Their code was written in an almost system independent manner. Some include directives have been changed in order to make them consistent with the unix file system. (I.e. a change of lower/upper-case letters.) Apart from this, the code of the C2A library has not been changed. C2A is based on the PQP library version 1.3. The version of C2A that has been used also carries the number 1.3.

The C2A library has been compiled with GCC 4.4.7.2 using the -O3 optimization option in combination with the NDEBUG compile constant. The same compile settings were used for the D2M system. Both systems were tested on a single thread ran by an Intel® Core™ i7-2670QM CPU @ 2.20GHz × 8 laptop having 6 GB of RAM. The operating system that was used was Ubuntu Linux 12.10.

The C2A library contains a small benchmarking program. The original intent was to run exactly the same benchmark. In their benchmark, two bunnies, each having 69,664 triangles are continuously pushed against each other. To be more precise, the benchmark consists out of approximately 300 frames. About 200 of them represent colliding contact, while 100 of them represent non-contact.

Each frame, consists out of a static bunny together with continuously translating and rotating bunny. Thus in their benchmark, only one bunny is actually moving, although one may get a different impression while watching the end result. In version 1.3 of their distribution, only one bunny is continuously moving. The other bunny changes orientation on a per-frame basis. The above seems to imply that they have been over-simplifying the problem. This is not the case. When making the rotation of the static bunny continuous, their system does not fail. The performance of their system only slightly decreases.

There is however another construct in their benchmark, that does raise many questions. Each frame is measured 10 times in a loop, presumably to achieve accuracy in time-measurement. The benchmark shows very very promising performance statistics. The performance of their system seems to be a bit exaggerated. The reason for this is that the number of tests is parametrizable. One would expect that a decent measurement of time is obtained after measuring 10 times. Though their time measurement seems to lower very significantly (exceeding a factor 2), when the number of trails per frame is set to e.g 1000. This means that the timer is either very inaccurate, or their is something wrong with their setup.

The computational efficiency of C2A seems to get significantly better when the number of trials is increased. The reason for this is not that obvious, but when closely examining their source code, it is clear that the trials are non-independent. After each trial, the closest pair of triangles is cached and used with the new iteration. This causes the average timing of one frame to converge

to approximately 1.6 milliseconds, which is much faster than the system would perform in the case of only one trial per frame.

Caching the closest triangle seems to be a good way to optimize the performance of a collision detection system in general when the collisions are coherent. But when the collisions are repeated in order to get a finer time-measurement, it is not appropriate. Because of this caching, their exact setup has been disqualified within the context of this thesis. When only timing once, their method still performs way better (in speed) at their own setup.

7.3 The Benchmarks used in this Thesis

There are 40 benchmarks that measure the computational efficiency and accuracy of D2M, D2M Capoeira and C2A. The tests have been constructed in such a way that the caching system of C2A is disabled as much as possible. This seems fair, though the caching system does serve a purpose. In reality, it is likely that a similar collision query can be performed multiple times (it depends on the application). Coherency between collisions is explicitly not tested within these benchmarks.

In order to make the tests as fair as possible, randomized tests are used, which are also used in the C2A paper. The main API function *C2A_Solve* was called in all benchmarks. This function returns a lot of data. The only value that was effectively used was the time of contact. This time was used to integrate the motions of both objects (using code associated with this thesis). Incidentally the distance between the two models, when using C2A reaches 0. In this context, 0 means that the surfaces are touching or that there is some penetration. The distances are measured with the *C2A_Distance* function that was included within the C2A library. The C2A distance function itself is based on the distance querying function that was supplied with the PQP (proximity query package) library.

Plots have shown that the motion of C2A is identical to the motion of D2M. This is also illustrated in their paper. Though in practice the type of motion is not identical at the floating point level.

Within the following context, the term random means pseudo random:

> All the benchmarks consists out of two objects being thrown against each other with linear translational and linear rotational motion. Both objects move along an axis. This axis is picked at random (i.e. a point on a uniformly distributed sphere). Both objects can initially be either 4 units apart, or they can initially be 512 units apart. Each frame simulates a collision that takes one second. Within this one second, both objects are moved to the origin, meaning that they both have a velocity of 2 units per second, or 256 units per second, yielding a relative velocity of 4 or 512 units per second. Additionally both objects also rotate. For each object, an axis of rotation is picked at random (again a point on a uniformly distributed sphere is picked). The objects are rotated along these random axis for an angle of 90 degrees. It is also required to mention that the initial orientation of both objects is also chosen at random. In order to do this, an element on the upper hemisphere of the unit quaternions is chosen uniformly at random.

The above described just one frame of a benchmark. Note that such a frame has a parameter, namely the kind of speed. The speed can be low i.e a 4 units per second collision, or it can be high, i.e. a 512 units per second collision (not considering the rotational velocity).

A single benchmark consists out of 256 high speed or low speed collisions. Each benchmarks represents either D2M against C2A or D2M Capoeira against C2A. The benchmarks involving D2M Capoeira happen to have odd index numbers.

Next to the speed, and whether Capoeira mode is used or not, the models may vary. Within the 40 benchmarks that show the difference between D2M and C2A, 4 models are used.

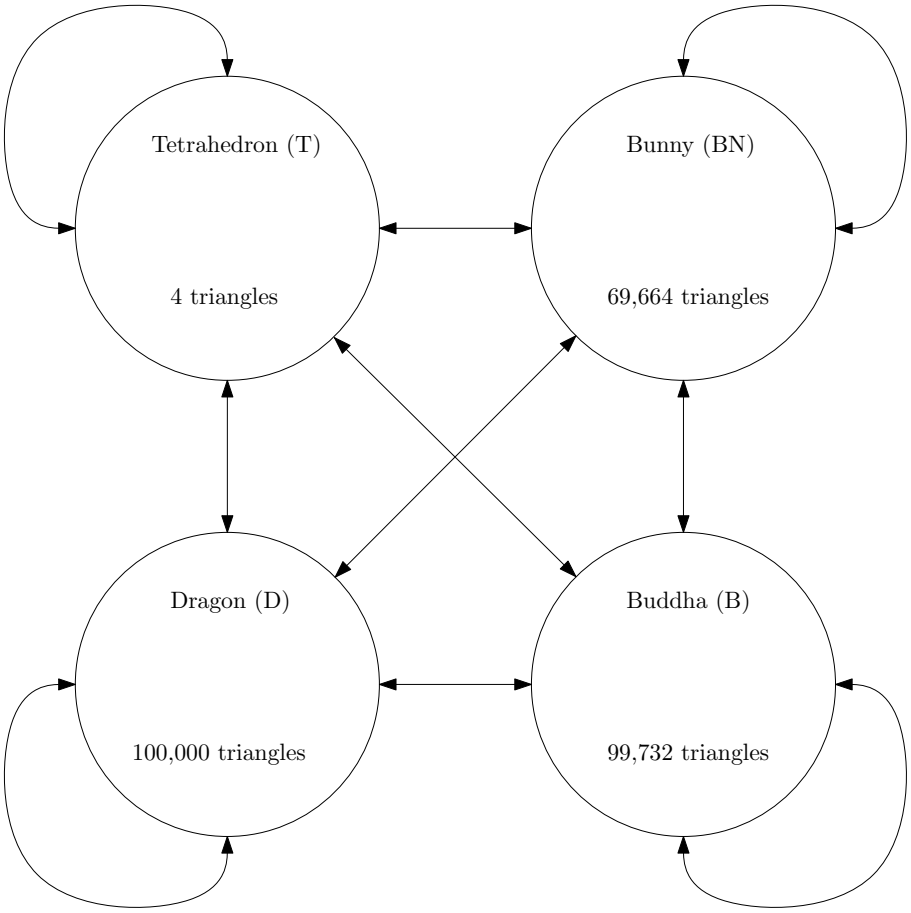

Figure 16: The experiment setup, each test (arrow) is performed 4 times.

As can be seen in Figure 16, each combination of two (possibly non-distinct) models is benchmarked. Each of these combinations is benchmarked four times. This is because there exists low speed/high speed benchmarks, and non-Capoeira mode/Capoeira mode benchmarks.

Also note the number of triangles of each model. In the bottom of the picture, the models contain approximately 100K triangles, while in the top of the picture, the models contain less. As an additional remark, it must be said

that all models are topologically closed. Meaning that from the outside the back of a triangle cannot be reached. This is not a requirement for the D2M method and neither is it a requirement of C2A.

It must be stated that in general, each frame of each benchmark is repeated 10 times, in order to improve timing accuracy. There is one exception though, the tetrahedron versus tetrahedron benchmarks are repeated 1000 times, in order to reduce the timing in-accuracy. With these tetrahedron benchmarks, it is even questionable whether the diagnostic message that is passed via the standard output slows down the process.

7.4 Highlighting the Benchmarks

When considering the benchmarks, the distances are measured using the C2A wrapper around the PQP library. This library seems to measure the distances with good precision. Contrary to the distances, the timings are inaccurate. This is because the actual time it takes to compute the benchmarks is measured by a clock. The scheduling of threads may cause the timings to be off by at most 4.0 milliseconds. For the tetrahedron versus tetrahedron benchmarks this is about 0.027 milliseconds (due to more precise timings).

7.4.1 The Low Speed Benchmarks

The concrete timing measurements for low speed collisions are summarized in Figure 17.

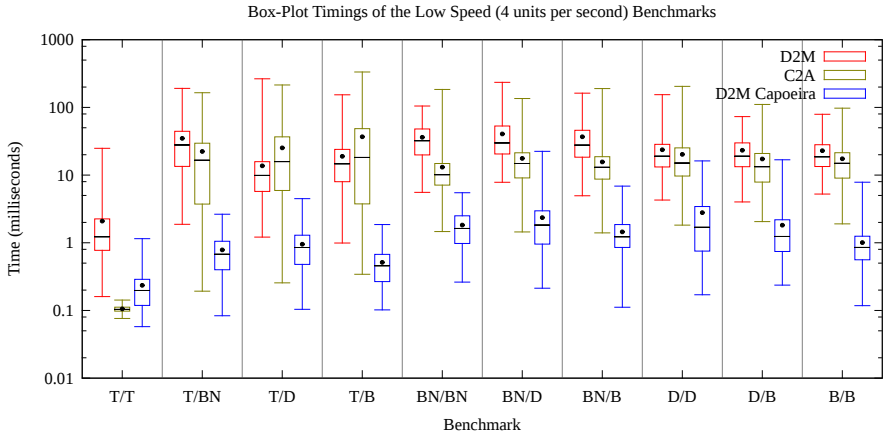

Figure 17

These box plots denote the minimum, the lower-quartile, the median (black stripe), the upper-quartile and the maximum time a collision instance took to compute within a single benchmark. It must be noted that the plot is on a logarithmic scale and that the dots represent the means of the distributions. As is clear from the box plot, the timings may vary a lot, both between different systems, but also between different collision instances of the same benchmark. Note that in the tetrahedron versus bunny (T/BN) benchmark, the timings for D2M may vary from 1.9 milliseconds, to $1.9 \cdot 10^2$ milliseconds. The box in the

middle is bounded by the lower and upper quartile. It can be stated that 75% of the timings does not exceed 44 milliseconds (tetrahedron versus bunny, D2M).

Note that in the tetrahedron versus tetrahedron benchmark, C2A performs outstanding in computational efficiency. In the worst case, it is about 170 times faster than D2M and about 8.1 times faster than D2M Capoeira. In the same benchmark, on average C2A is about 20 times faster than D2M and 2.2 times faster than D2M Capoeira.

Depending on the type of application, one may be interested in a different kind of statistic.

All three systems are described using various statistics. If you want to use the systems in their current form for e.g. a real-time simulation, then you do need to look at the worst case performance. *The worst cases look bad for D2M as well as for C2A.*

All collision instances of a benchmark are similar in conceptual difficulty. Considering this, it does not seem to make much sense that there is such a huge variation in timings. Though both D2M and C2A display this behaviour.

When one wishes to use these collision detection systems in a non-time critical context, one would probably want to look at the mean timings (indicated with the dot). Because the logarithmic scale can be quite confusing, the statistics have been included in tables in Appendix K. Figure 17 does not stand by itself. It is accompanied by a different figure, namely a box-plot of distances between the objects at the TOC of the associated collision instance (see Figure 18).

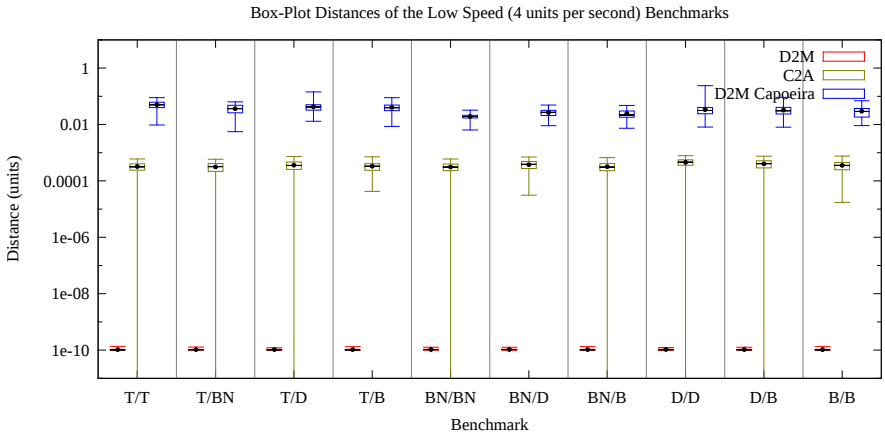

Figure 18

When reviewing the picture, one immediately notices the high accuracy of D2M. What is also quite visible is that the minimum distance of C2A reaches below 10^{-11} in various instances. In these cases, the diagnostic message of C2A that is printed after each collision indicates that the distance is 0. Thus both the distance measurement used to obtain the data and the diagnostic message of C2A correspond. A distance of 0 either means that the system is penetrating or touching. The latter is unwanted for the purpose of simulation.

However, probably this has to do with degenerate cases of C2A. In all benchmarks C2A touches or penetrates in only 15 of the 5120 collision instances. To

be honest, during the development of the D2M system, the benchmarks failed on numerous occasions (measuring a distance of 0). By adjusting the system repeatedly, the benchmarks eventually reached a perfect (penetration) score of 0 (both D2M and D2M Capoeira). This does in no way mean that the system is flawless in that sense. If someone were to create a new benchmark, this benchmark could either cause penetration or maybe degrade the performance of the system to unknown depths. Concluding the text above; that C2A penetrates incidentally does not degrade their system from a theoretical point of view.

Additionally it must be noted that in D2M Capoeira mode, the distances are measured between the models and not between the bounding volume hierarchy around the models. Although the simulation results of D2M do not suggest this, it may be the case that the hierarchies are penetrating incidentally, thus making simulation less robust. The latter has not been tested in a decent manner.

In general, when considering the worst-case distances, D2M is about a million times more accurate than C2A *in its default configuration*. In the tetrahedron versus tetrahedron case, the worst-case timing difference between D2M and C2A is extreme though (about a factor 170 in the advantage of C2A). Still, if one prefers accuracy over performance, D2M is still a valid option.

With D2M, the accuracy is parametrizable, though setting the accuracy to low does not improve the performance significantly. This is due to the fast convergence of the hybrid root finder that was used. With the C2A code that was supplied, the accuracy had been fixed. Without changing the source code, a way has been found to vary the accuracy of C2A. And that is by varying the speed of the objects.

Before getting into the high-speed benchmarks, it can be stated that in general, the performance of D2M is comparable to the performance of C2A though slightly worse, with the exception of the tetrahedron versus tetrahedron benchmark in which C2A excels in performance. In any case D2M is about a million times more accurate.

D2M Capoeira seems to be faster than C2A, but the accuracy of D2M Capoeira is rather disappointing at low speeds. C2A is about a hundred times more accurate than D2M Capoeira at low speeds. This accuracy difference is rather significant because it is even visible when looking at both objects as a whole. In its worst collision instance, D2M Capoeira has a collision distance of 0.24. This is actually noticeable since the dragon has a diameter of about 3.5 units.

For the purpose of comparing relative accuracy, this should be done by considering the worst cases.

7.4.2 The High Speed Benchmarks

The high speed benchmarks are summarized in Figures 19 and 20.

The reason why these benchmarks have been included is because it makes it possible to compare D2M Capoeira against C2A in accuracy. The timings of C2A against D2M are no match, because the worst case accuracy of D2M is always better than 10^{-9}, while the worst case accuracy of D2M Capoeira and C2A is almost comparable around 10^{-1} (C2A is significantly more precise though). It is almost possible to discriminate between C2A and D2M Capoeira considering these rather extreme benchmarks by just looking at the timings.

Another reason why these extreme benchmarks have been included is to show what the systems are capable of. If one considers the units to be meters, then both objects are moving at near super-sonic speeds. The combined relative velocity would be 512 meters per second (not considering the rotations), which is super-sonic. It must be noted that within this context, D2M is way too accurate for practical purposes. Yet, if one can spare the processing power, there is little reason not to use it.

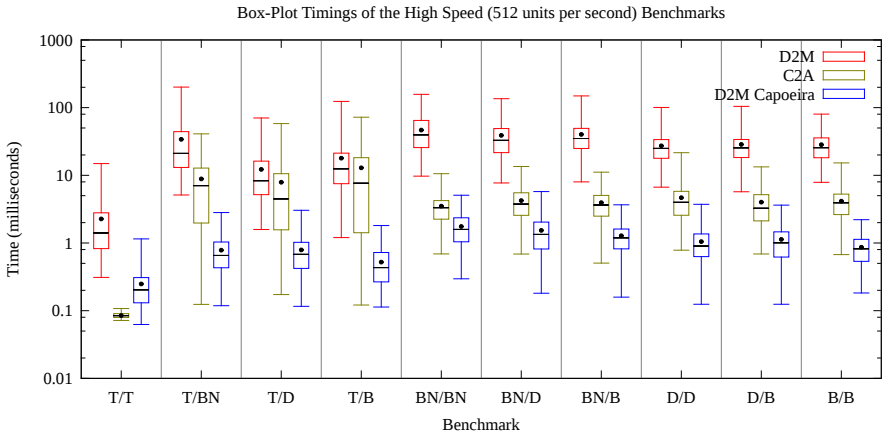

Figure 19

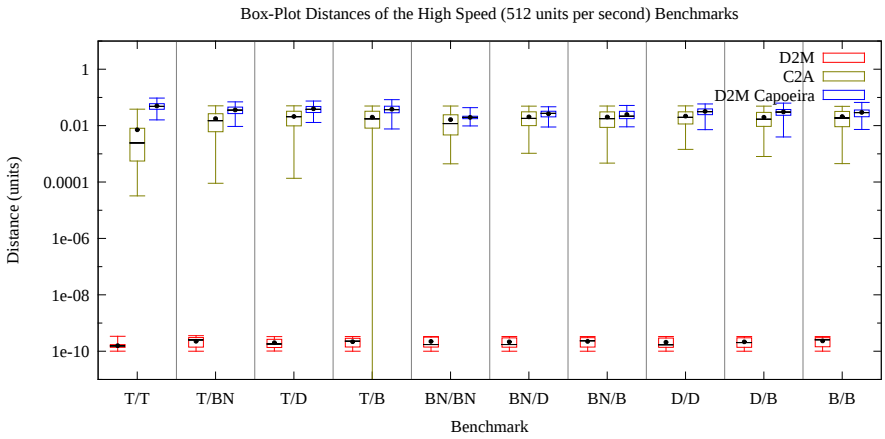

Figure 20

Again this time C2A excels at the tetrahedron versus tetrahedron case. This benchmark also illustrates the weakness of D2M Capoeira, meaning that only the bounding volume hierarchy gets collided and not the actual geometry inside. Again it must be noted that D2M Capoeira is a very approximate form of continuous collision detection. Though in the other benchmarks it does not seem to do much worse than C2A when looking at (high-speed) accuracy. In fact, D2M Capoeira displays more stable results than C2A (considering the high-

speed benchmarks). Also, D2M Capoera is faster than C2A in the (high-speed) non tetrahedron versus tetrahedron cases.

7.5 The Verdict

It is possible to aggregate over all the 40 benchmarks. The differences between the methods can be summarized by Figures 21 and 22. Again both tables are plotted on a logarithmic scale:

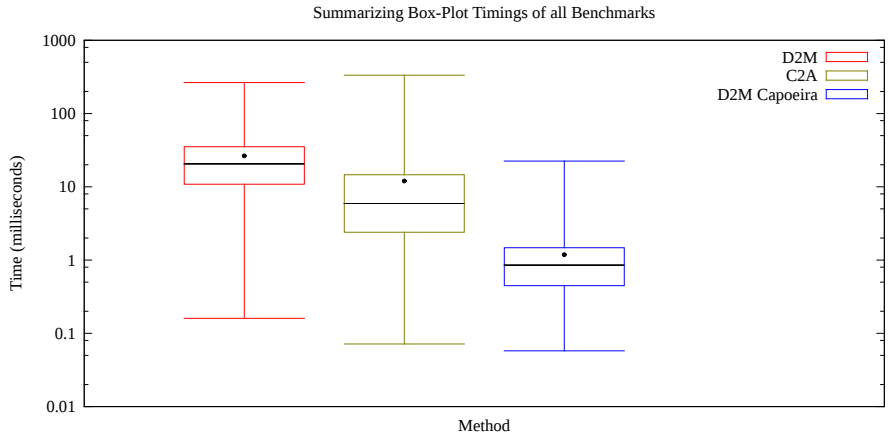

Figure 21

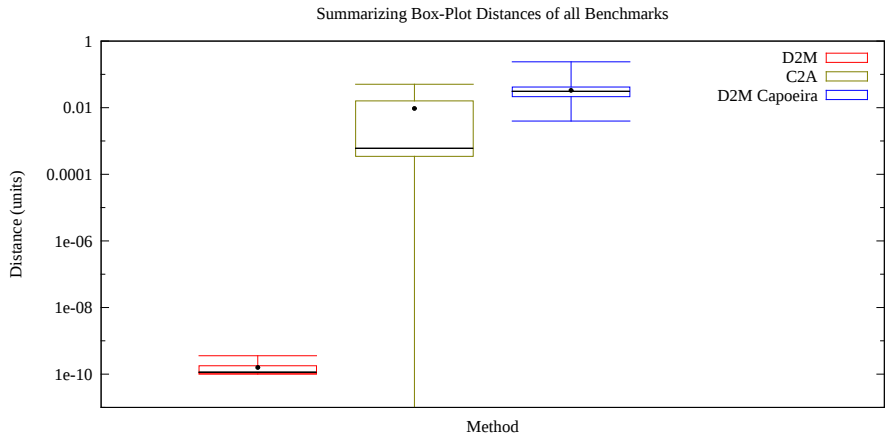

Figure 22

The numbers above are also listed in a table in Appendix K.21. The data above summarizes all 5120 collision instances. Considering these benchmarks, D2M Capoeira is the fastest but the least accurate. D2M excels in accuracy but it is less computationally efficient than C2A.

By decreasing the accuracy of D2M, one does not get a considerable improvement in speed.

8 Future Work / Discussion

Truly continuous collision detection is still a research branch. The system presented in this thesis seems to do quite well. Yet if you combine it with an unproven numerical collision response algorithm, you get something that works in most cases, but not always. Thus, a field of study that remains is how to handle collision responses 100% of the time. Also more attention can be added to sustaining collisions.

Non-penetration is essential for many systems that involve C.C.D.. The benchmarks suggest that the D2M / D2M Capoeira systems are indeed non-penetrating. However, the simulation part fails on incidental occasions. To be thorough, the precise C.C.D. system seems to fail on rare occasions when embedded into a simulation. Sometimes, a distance of 0 is measured (within the simulation part). Let us just state that the system (D2M) has its weaknesses. On the other hand, Capoeira mode seems to be stable enough to be used for simulation purposes. Yet, parts in the source code involving Capoeira mode do not make much sense. The parts mentioned within this paragraph require some work.

Computer simulations demand high performance of its sub-components. The construction time of the custom BVH can be reduced. By cutting away excessive constructions, it may also be possible to optimize the system even further.

The projects associated with this thesis use a customized implementation of interval-arithmetic. This implementation does not account for floating point roundings. Using an interval arithmetic library that does account for this would be a good thing when considering robustness. When considering ease of implementation and speed, the current interval arithmetic implementation is doing just fine.

In theory the motion can be customized. The latter requires some derivations involving analytical derivatives to be done again. The current derivations in the appendix are often simplified due to a property of linear translational and linear rotational motion. An essential property of any type of motion is that it can be cut into pieces along the time axis, and that each of these pieces is of the same motion type as the big piece (which was cut).

As stated earlier, the system does not handle big flat surfaces that collide with tiny triangles in a decent manner. This issue needs to be tackled. There are solutions to this problem, but within the context of this thesis there is no time to solve them.

Now, we have polygon-object A versus polygon-object B. Perhaps it would be a good thing to also allow collisions with spheres, or static planes/polygons. This would make the system even more complex, but in certain contexts, it is not preferred to triangulate a sphere. Likewise, if the geometry is static (or non-rotating), bounding boxes may do a better job than bounding spheres.

Issues concerning scalability have not been discussed. It is always possible to put different collisions on different threads. The BVHs are read-only, so it will not cause any read-write concurrency problems. A problem that arises is that there does not seem to be a way of knowing which collisions will be heavy and which of them will be light, considering their computational load. During the coarse of the project, no efficient solution was found to tackle this problem.

Last, but not least, resting contact may also shrink the computational demand put on the C.C.D. system. If object B is hovering over object A, then this

reduces the strain on the C.C.D. system. Otherwise, object B and object A, might collide e.g. 32 times per second. The latter is likely to stop the simulation from performing in real-time.

9 Conclusion

During the span of this master-project, the collision detection problem as it is described in the introduction, has been solved in both 2D and 3D. The 3D system (D2M) has been benchmarked next to a competing system; C2A. In general C2A is faster, but D2M is very-much more accurate. When two object collide with D2M, the distance in-between objects is within the range $[1, 4] \cdot 10^{-10}$.

The D2M system uses a technique that is based on Taylor's theorem (with the Lagrange remainder) in combination with the bisection of intervals. When this technique is compared with the more simplistic interval-bisection, the more complicated technique is about 6 or 7 times faster than bisection on intervals.

The data structure that was used in order to partition the model is an R-tree. Each node of the R-tree also stores a disk/sphere to bound the geometry inside. The latter is useful when the objects rotate. A lot of effort has been invested in finding the right broad-phase continuous collision primitives. The primitive that was eventually found (approximate sphere/sphere test) fits the eventual BVH quite well . The current impression is that the current Hierarchy is "good but not great". As written in the related work section, the hierarchy could be improved upon by just considering more literature.

As a by-product, the D2M Capoeira method was found; a very approximate form of C.C.D. . D2M Capoeira performs good enough under certain circumstances and it is much faster than D2M in general.

To conclude positively, the system (D2M) presented in this thesis is highly accurate, and its computational performance is comparable to a system that is considered to be state of the art.

References

[1] Robert A. Adams. *Calculus, A Complete Course, Sixth edition.* Pearson Addison Wesley, 2006.

[2] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *In Proc. of ACM SIGGRAPH 89*, pages 223–232, 1989.

[3] David Baraff. Physically based modeling rigid body simulation. 2001.

[4] Jan Bender and Alfred Schmitt. Constraint-based collision and contact handling using impulses. 2006.

[5] Stephen Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, 6:291–302, 1990.

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

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

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

[9] Erik B. Dam, Martin Koch, and Martin Lillholm. Quaternions, interpolalation and animation. 1998.

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

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

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

[13] Berthold Bäuml Holger Täubig and Udo Frese. Real-time swept volume and distance computation for self collision detection. 2011.

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

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

[16] 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.

[17] Mark H. Overmars. Efficient data structures for range searching on a grid, 1987.

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

[19] 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.

[20] 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.

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

[22] Gino van den Bergen. *Collision Detection in Interactive 3D Computer Animation*. University Press, 1999.

[23] Gino van den Bergen. Ray casting against general convex objects with application to continuous collision detection. 2004.

[24] Xinyu Zhang, Minkyoung Lee, and Young J. Kim. Interactive continuous collision detection for non-convex polyhedra. *Vis. Comput.*, 22(9):749–760, September 2006.

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

Appendix

A Linear Translational and Linear Rotational Motion

In this section, the concept of linear translational and linear rotational motion will be mathematically defined in both 2D and 3D.

A.1 Linear Translational and Linear Rotational Motion in 2D

Let a linear translational and linear rotational motion in 2D be described by a 4-tuple: $(\mathbf{o}_0, \mathbf{v}, \alpha_0, \omega) \in \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R} \times \mathbb{R}$.

Let $\mathbf{o}_0$ be the origin of the object at $t = 0$ and let $\mathbf{v}$ be the translational velocity along the entire time domain. Let α_0 be the initial orientation (at $t = 0$), measured in counter clockwise radians with respect to the positive x-axis. Let ω be the rotational velocity in counter clockwise radians per time unit along the entire time domain.

The concept is described best as a transformation from local object-space to world-space by using the following 3×3 parametrized homogeneous transformation matrix ($\mathbf{M}$):

$$\mathbf{R}(x) = \begin{pmatrix} \cos(x) & -\sin(x) & 0 \\ \sin(x) & \cos(x) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{15}$$

$$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} 1 & 0 & x_1 \\ 0 & 1 & x_2 \\ 0 & 0 & 1 \end{pmatrix} \tag{16}$$

$$\mathbf{M}(t) = \mathbf{T}(\mathbf{o}_0 + \mathbf{v} \cdot t) \cdot \mathbf{R}(\alpha_0 + \omega \cdot t) \tag{17}$$

A.2 Linear Translational and Linear Rotational Motion in 3D

Let a linear translational and linear rotational motion in 3D be described by a 4-tuple: $(\mathbf{o}_0, \mathbf{v}, \mathbf{q}_0, \omega) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{H} \times \mathbb{R}^3$. Similar to the 2D case, let $\mathbf{o}_0$ be the origin of the object at $t = 0$ and let $\mathbf{v}$ be the translational velocity of the object along the entire time domain. Let $\mathbf{q}_0$ be a unit quaternion that represents the initial orientation of the object (i.e. at $t = 0$). Let ω be a radial vector that represents the rotational velocity of the object along the entire time domain.

A radial vector has a direction and a magnitude. Let the axis be the axis of rotation $\mathbf{a} = \frac{\omega}{\|\omega\|_2}$, and let the magnitude (i.e. $\|\omega\|_2$) be the number of radians per time unit that the object is rotating in counter-clockwise direction (when the axis is pointing into the direction of the observer).

This rotation can be converted into (unit) quaternion form by using the formula $\mathbf{s}(\beta) = \sin\left(\frac{\beta}{2}\right) \begin{pmatrix} i \\ j \\ k \end{pmatrix} \cdot \mathbf{a} + \cos\left(\frac{\beta}{2}\right)$, where β is the amount of rotation along $\mathbf{a}$ in counter clockwise radians (when $\mathbf{a}$ is pointing into the direction of

the observer). Note that this formula is not well defined when $\omega = \mathbf{0}$. In this case, $\mathbf{a}$ is not defined, and the way to correctly deal with the situation would be to make $\mathbf{s}(\beta)$ equal to 1 (the identity quaternion).

Now we can define linear translational and linear rotational motion as a function of time to pair: $\text{ltlrm} : \mathbb{R} \mapsto \mathbb{H} \times \mathbb{R}^3$.

$$\text{ltlrm}(t) = (\mathbf{s}(\|\omega\|_2 \cdot t) \cdot \mathbf{q}_0, \mathbf{o}_0 + \mathbf{v} \cdot t) \tag{18}$$

The parametrized pair above gives us a rotation followed by a translation. This is conceptually identical to the matrix definition given for 2D linear translational and linear rotational motion above. Nevertheless, it is possible to be a little more explicit.

Let the rotation of point $\mathbf{p} \in \mathbb{H}$ by a unit quaternion $\mathbf{x} \in \mathbb{H}$, be defined by:

$$\mathbf{r}(\mathbf{x}, \mathbf{p}) = \mathbf{x} \cdot \mathbf{p} \cdot \mathbf{x}^*. \tag{19}$$

Where $\mathbf{x}^*$ is the conjugate quaterion of $\mathbf{x}$. The quaternion $\mathbf{p}$ is defined as $i \cdot u_1 + j \cdot u_2 + k \cdot u_3$, where $\mathbf{u} = (u_1, u_2, u_3)^T$ represents the 3D point in the initial, non-rotated space.

Now we can use Equation 19 to construct a mapping from local space to world space:

$$\mathbf{m}(\mathbf{p}, t) = \mathbf{r}(\text{ltlrm}_1(t), \mathbf{p}) + \begin{pmatrix} i \\ j \\ k \end{pmatrix} \cdot \text{ltlrm}_2(t). \tag{20}$$

The mapping above is invertible:

$$\mathbf{m}^{-1}(\mathbf{p}, t) = \mathbf{r}\left(\text{ltlrm}_1^*(t), \mathbf{p} - \begin{pmatrix} i \\ j \\ k \end{pmatrix} \cdot \text{ltlrm}_2(t)\right). \tag{21}$$

B Derivatives

This appendix describes the computation of the derivatives that are used within the thesis.

B.1 Derivatives of a Point under Linear Translational and Linear Rotational Motion

Given a point in world-space, that is located on/in an object, compute its derivative with respect to time. When this derivative has been described, the second derivative with respect to time will be derived.

B.1.1 The 2D case

We define $\mathbf{r}(t)$ as $\mathbf{p}(t) - \mathbf{o}(t)$ where $\mathbf{p}(t)$ is the point defined in world-space of which we want to know the velocity and $\mathbf{o}(t)$ is the origin of the object. The velocity of a point in 3D is conveniently defined by a formula.

$$\frac{d\mathbf{p}(t)}{dt} = \omega \times \mathbf{r}(t) + \mathbf{v} \tag{22}$$

The 2D analogy of this equation is:

$$\frac{d\mathbf{p}(t)}{dt} = \omega \begin{pmatrix} -r_2(t) \\ r_1(t) \end{pmatrix} + \mathbf{v} \tag{23}$$

Note that $\begin{pmatrix} -r_2(t) \\ r_1(t) \end{pmatrix}$ describes $\mathbf{r}(t)$, but then rotated 90 degrees counter-clockwise. Thus $\begin{pmatrix} -r_2(t) \\ r_1(t) \end{pmatrix}$ is perpendicular to $\mathbf{r}(t)$, and of the same magnitude. Suppose the object would have a velocity of $\mathbf{0}$, and an angular velocity of 1, then $\begin{pmatrix} -r_2(t) \\ r_1(t) \end{pmatrix}$ would be the velocity of point $\mathbf{p}(t)$. The angular velocity can be varied, and a linear translational component can be added.

Now we describe the second derivative of $\mathbf{p}(t)$.

$$\frac{d^2\mathbf{p}(t)}{dt^2} = \omega \begin{pmatrix} -\frac{dr_2(t)}{dt} \\ \frac{dr_1(t)}{dt} \end{pmatrix} \tag{24}$$

Where $\frac{d\mathbf{r}(t)}{dt} = \frac{d\mathbf{p}(t)}{dt} - \frac{d\mathbf{o}(t)}{dt} = \frac{d\mathbf{p}(t)}{dt} - \mathbf{v}$. Note that additional simplifications are possible.

B.1.2 The 3D case

The first derivative of a point is basically known (see Equation 22). The second derivative can be found by using the following identity[5]:

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".

$\frac{d}{dx}(\mathbf{a} \times \mathbf{b}) = \frac{d\mathbf{a}}{dx} \times \mathbf{b} + \mathbf{a} \times \frac{d\mathbf{b}}{dx}$

Applying $\frac{d}{dt}$ on equation 22, yields:

$\frac{d^2\mathbf{p}(t)}{dt^2} = \frac{d}{dt}(\omega \times \mathbf{r}(t)) + \frac{d\mathbf{v}}{dt}$
$\frac{d^2\mathbf{p}(t)}{dt^2} = \frac{d\omega}{dt} \times \mathbf{r}(t) + \omega \times \frac{d\mathbf{r}(t)}{dt}$
$\frac{d^2\mathbf{p}(t)}{dt^2} = \omega \times \frac{d\mathbf{r}(t)}{dt}$

The resultant equation is:

$$\frac{d^2\mathbf{p}(t)}{dt^2} = \omega \times (\omega \times \mathbf{r}(t)) \tag{25}$$

B.2 The Derivatives of Distance Functions

In this section, derivatives will be derived of distance functions.

B.2.1 A Line and a Point

This function is only required in 2D, since vertex/edge collisions are not applicable in 3D. The function will thus be defined in 2D. Let $\mathbf{a}, \mathbf{b}, \delta \in \mathbb{R}^2$ (see Figure 23).

[5]This product rule is literally quoted from Wikipedia.

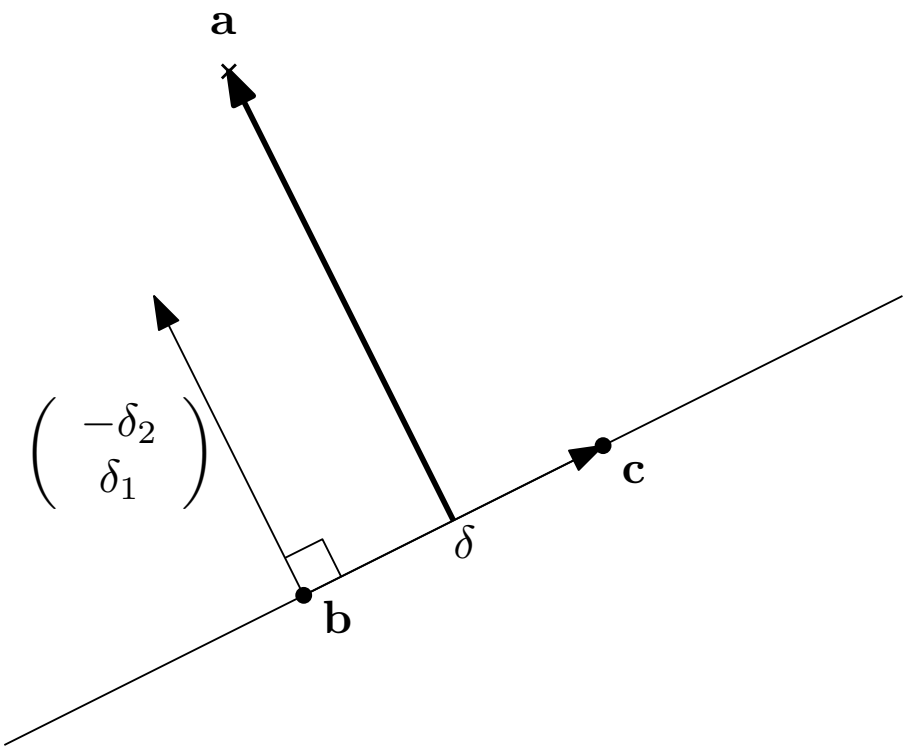

Figure 23: The distance between a point and a line.

Note that the point **a** is located on object A, and that the points **b** and **c** are located on object B. The vector δ is defined as $\mathbf{c} - \mathbf{b}$. Let us start by defining the 0^{th} derivative of the distance function, i.e. the distance function itself. Let $\|\delta\|_2$ be a constant, let $\gamma(t)$ be $\mathbf{a}(t) - \mathbf{b}(t)$ and let $\delta^*(t)$ be $\begin{pmatrix} -\delta_2(t) \\ \delta_1(t) \end{pmatrix}$ then:

$$s(t) = \frac{1}{\|\delta\|_2}\delta^*(t) \cdot \gamma(t). \tag{26}$$

Where $s(t)$ is the distance toward the edge. By using the product rule, we can take the derivative:

$$\frac{ds(t)}{dt} = \frac{1}{\|\delta\|_2}\left(\frac{d\delta^*(t)}{dt} \cdot \gamma(t) + \delta^*(t) \cdot \frac{d\gamma(t)}{dt}\right). \tag{27}$$

If we take the derivative again (by again using the product rule and simplifying afterwards), we get:

$$\frac{d^2s(t)}{dt^2} = \frac{1}{\|\delta\|_2}\left(\frac{d^2\delta^*(t)}{dt^2} \cdot \gamma(t) + 2\frac{d\delta^*(t)}{dt} \cdot \frac{d\gamma(t)}{dt} + \delta^*(t) \cdot \frac{d^2\gamma(t)}{dt^2}\right). \tag{28}$$

The formulas are deliberately kept in a non-explicit form. It is preferable to keep the notation as compact as possible, because when the derivatives are evaluated, they are evaluated using functions on vectors instead of scalars.

B.2.2 The Signed Distance between two Spheres

Let there be two spheres, sphere A and sphere B. Let $\mathbf{a}(t) \in \mathbb{R}^3$ be the origin of sphere A and let $\mathbf{b}(t) \in \mathbb{R}^3$ 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 $\gamma(t)$ be $\mathbf{b}(t) - \mathbf{a}(t)$. If $\|\gamma(t)\|_2 = r_A + r_B$ then the surfaces of the spheres are touching.

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

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

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 $s(t) = 0$:

$$s(t) = \gamma(t) \cdot \gamma(t) - (r_A + r_B)^2. \tag{29}$$

Again we derive:

$$\frac{ds(t)}{dt} = \frac{d\gamma(t)}{dt} \cdot \gamma(t) + \gamma(t) \cdot \frac{d\gamma(t)}{dt} = 2\frac{d\gamma(t)}{dt} \cdot \gamma(t). \tag{30}$$

and

$$\frac{d^2 s(t)}{dt^2} = 2\left(\frac{d^2\gamma(t)}{dt^2} \cdot \gamma(t) + \frac{d\gamma(t)}{dt} \cdot \frac{d\gamma(t)}{dt}\right). \tag{31}$$

B.2.3 Two Lines (in 3D)

Let there be 4 points in world-space: $\mathbf{a}(t)$, $\mathbf{b}(t)$, $\mathbf{c}(t)$ and $\mathbf{d}(t)$, and let $(\mathbf{a}(t), \mathbf{b}(t))$ and $(\mathbf{c}(t), \mathbf{d}(t))$ define ℓ_1 and ℓ_2 respectively. Let $\delta_1(t)$ be $\mathbf{b}(t) - \mathbf{a}(t)$ and let $\delta_2(t)$ be $\mathbf{d}(t) - \mathbf{c}(t)$. Let $\mathbf{n}(t) = \delta_1(t) \times \delta_2(t)$, be the direction in which the distance between the lines segments should be measured. The latter statement is only true when the lines are non-parallel. If the lines are parallel, $\mathbf{n}(t) = 0$. In the case of non-parallel lines, the signed distance $s(t)$ equals $f(t) \cdot \mathbf{n}(t) \cdot (\mathbf{c}(t) - \mathbf{a}(t))$ for some strictly positive scalar function $f(t)$. Typically, if one is interested in the distance for the non-parallel lines, one would pick $f(t) = \frac{1}{\|\mathbf{n}(t)\|_2}$. However we are not interested in the distance but in the roots of the distance function. By leaving out the normalization, we also find the intersections of both lines. The function $s(t)$ also evaluates to 0 when lines are parallel.

Thus, the roots lie at line intersections and they lie at moments in time where the lines are parallel. When the roots are eventually determined, a static check has to show whether there is indeed a line/line intersection, or ultimately an edge/edge intersection.

$$s(t) = \mathbf{n}(t) \cdot \gamma(t). \tag{32}$$

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

$$\frac{ds(t)}{dt} = \frac{d\mathbf{n}(t)}{dt} \cdot \gamma(t) + \mathbf{n}(t) \cdot \frac{d\gamma(t)}{dt}. \tag{33}$$

Let us now derive $\frac{d\mathbf{n}(t)}{dt}$:

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

$$\frac{d\mathbf{n}(t)}{dt} = \frac{d\delta_1(t)}{dt} \times \delta_2(t) + \delta_1(t) \times \frac{d\delta_2(t)}{dt}. \tag{34}$$

It is of course also required to state $\frac{d\delta_1(t)}{dt}$, $\frac{d\delta_2(t)}{dt}$ and $\frac{d\gamma(t)}{dt}$:

$$\frac{d\delta_1(t)}{dt} = \frac{d\mathbf{b}(t)}{dt} - \frac{d\mathbf{a}(t)}{dt}. \tag{35}$$
$$\frac{d\delta_2(t)}{dt} = \frac{d\mathbf{d}(t)}{dt} - \frac{d\mathbf{c}(t)}{dt}. \tag{36}$$
$$\frac{d\gamma(t)}{dt} = \frac{d\mathbf{c}(t)}{dt} - \frac{d\mathbf{a}(t)}{dt}. \tag{37}$$

Applying $\frac{d}{dt}$ on equation 33 yields:

$\frac{d^2s(t)}{dt^2} = \frac{d}{dt}\left(\frac{d\mathbf{n}(t)}{dt} \cdot \gamma(t)\right) + \frac{d}{dt}\left(\mathbf{n}(t) \cdot \frac{d\gamma(t)}{dt}\right)$.
$\frac{d^2s(t)}{dt^2} = \frac{d^2\mathbf{n}(t)}{dt^2} \cdot \gamma(t) + \frac{d\mathbf{n}(t)}{dt} \cdot \frac{d\gamma(t)}{dt} + \frac{d\mathbf{n}(t)}{dt} \cdot \frac{d\gamma(t)}{dt} + \mathbf{n}(t) \cdot \frac{d^2\gamma(t)}{dt^2}$.

$$\frac{d^2s(t)}{dt^2} = \frac{d^2\mathbf{n}(t)}{dt^2} \cdot \gamma(t) + 2\frac{d\mathbf{n}(t)}{dt} \cdot \frac{d\gamma(t)}{dt} + \mathbf{n}(t) \cdot \frac{d^2\gamma(t)}{dt^2}. \tag{38}$$

Where $\frac{d^2\mathbf{n}(t)}{dt^2}$ is derived as follows:

$\frac{d^2\mathbf{n}(t)}{dt^2} = \frac{d}{dt}\left(\frac{d\delta_1(t)}{dt} \times \delta_2(t) + \delta_1(t) \times \frac{d\delta_2(t)}{dt}\right)$.
$\frac{d^2\mathbf{n}(t)}{dt^2} = \frac{d}{dt}\left(\frac{d\delta_1(t)}{dt} \times \delta_2(t)\right) + \frac{d}{dt}\left(\delta_1(t) \times \frac{d\delta_2(t)}{dt}\right)$.
$\frac{d^2\mathbf{n}(t)}{dt^2} = \frac{d^2\delta_1(t)}{dt^2} \times \delta_2(t) + \frac{d\delta_1(t)}{dt} \times \frac{d\delta_2(t)}{dt} + \frac{d\delta_1(t)}{dt} \times \frac{d\delta_2(t)}{dt} + \delta_1(t) \times \frac{d^2\delta_2(t)}{dt^2}$.

Resulting in:

$$\frac{d^2\mathbf{n}(t)}{dt^2} = \frac{d^2\delta_1(t)}{dt^2} \times \delta_2(t) + 2\frac{d\delta_1(t)}{dt} \times \frac{d\delta_2(t)}{dt} + \delta_1(t) \times \frac{d^2\delta_2(t)}{dt^2}. \tag{39}$$

For completeness, it is also required to mention $\frac{d^2\gamma(t)}{dt^2}$, $\frac{d^2\delta_1(t)}{dt^2}$ and $\frac{d^2\delta_2(t)}{dt^2}$:

$$\frac{d^2\gamma(t)}{dt^2} = \frac{d^2\mathbf{c}(t)}{dt^2} - \frac{d^2\mathbf{a}(t)}{dt^2}. \tag{40}$$
$$\frac{d^2\delta_1(t)}{dt^2} = \frac{d^2\mathbf{b}(t)}{dt^2} - \frac{d^2\mathbf{a}(t)}{dt^2}. \tag{41}$$
$$\frac{d^2\delta_2(t)}{dt^2} = \frac{d^2\mathbf{d}(t)}{dt^2} - \frac{d^2\mathbf{c}(t)}{dt^2}. \tag{42}$$

B.2.4 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 $\mathbf{a}$ be our point in A-space. Let $\mathbf{b}$ be a point on the plane in B-space, and let $\mathbf{c}$ be a point in front of the plane, i.e. $\mathbf{c} = \mathbf{b} + \hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is the normal of the plane. Now that we have defined $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ 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 $\mathbf{a}(t)$ and the plane in world-space:

$$s(t) = \gamma(t) \cdot \hat{\mathbf{n}}(t). \tag{43}$$

Where $\gamma(t) = \mathbf{a}(t) - \mathbf{b}(t)$ and $\hat{\mathbf{n}}(t) = \mathbf{c}(t) - \mathbf{b}(t)$. Applying $\frac{d}{dt}$ on equation 43, yields:

$\frac{ds(t)}{dt} = \frac{d}{dt}(\gamma(t) \cdot \hat{\mathbf{n}}(t))$

$$\frac{ds(t)}{dt} = \frac{d\gamma(t)}{dt} \cdot \hat{\mathbf{n}}(t) + \gamma(t) \cdot \frac{d\hat{\mathbf{n}}(t)}{dt}. \tag{44}$$

Were $\frac{d\gamma(t)}{dt} = \frac{d\mathbf{a}(t)}{dt} - \frac{d\mathbf{b}(t)}{dt}$ and $\frac{d\hat{\mathbf{n}}(t)}{dt} = \frac{d\mathbf{c}(t)}{dt} - \frac{d\mathbf{b}(t)}{dt}$.

Now, we take the derivative once more:

$\frac{d^2s(t)}{dt^2} = \frac{d}{dt}\left(\frac{d\gamma(t)}{dt} \cdot \hat{\mathbf{n}}(t) + \gamma(t) \cdot \frac{d\hat{\mathbf{n}}(t)}{dt}\right)$
$\frac{d^2s(t)}{dt^2} = \frac{d}{dt}\left(\frac{d\gamma(t)}{dt} \cdot \hat{\mathbf{n}}(t)\right) + \frac{d}{dt}\left(\gamma(t) \cdot \frac{d\hat{\mathbf{n}}(t)}{dt}\right)$
$\frac{d^2s(t)}{dt^2} = \left(\frac{d^2\gamma(t)}{dt^2} \cdot \hat{\mathbf{n}}(t) + \frac{d\gamma(t)}{dt} \cdot \frac{d\hat{\mathbf{n}}(t)}{dt}\right) + \left(\frac{d\gamma(t)}{dt} \cdot \frac{d\hat{\mathbf{n}}(t)}{dt} + \gamma(t) \cdot \frac{d^2\hat{\mathbf{n}}(t)}{dt^2}\right)$

Resulting in:

$$\frac{d^2s(t)}{dt^2} = \frac{d^2\gamma(t)}{dt^2} \cdot \hat{\mathbf{n}}(t) + 2\frac{d\gamma(t)}{dt} \cdot \frac{d\hat{\mathbf{n}}(t)}{dt} + \gamma(t) \cdot \frac{d^2\hat{\mathbf{n}}(t)}{dt^2} \tag{45}$$

Were $\frac{d^2\gamma(t)}{dt^2} = \frac{d^2\mathbf{a}(t)}{dt^2} - \frac{d^2\mathbf{b}(t)}{dt^2}$ and $\frac{d^2\hat{\mathbf{n}}(t)}{dt^2} = \frac{d^2\mathbf{c}(t)}{dt^2} - \frac{d^2\mathbf{b}(t)}{dt^2}$.

C Fuzzy Line Segment Intersection

When one considers a trajectory of a point that corresponds with a time interval, it is possible to put a bounding box around this trajectory such that the location of the point is bounded. This box is equivalent to an axis aligned bounding box, and can be computed with the help of interval arithmetic.

If we have two line segments, we have four vertices. Since we sample the line segments over a time interval, each vertex has a trajectory associated with it. The trajectory of each vertex is bounded by an axis aligned bounding box.

Next, a capsule is drawn around each box pair that resembles an edge (see Figure 24).

This capsule can be found quite easily. When given one box, one can easily compute the centroid. It is possible to construct a minimal sphere around the box, by using basic math. Such a sphere has a radius. Note that there are two vertices for each edge. Both of them have bounding spheres. We define the radius of the capsule to be the largest of the radii of both boxes/vertexes.

These edges may or may not touch each other. By looking just at the boxes derived with interval arithmetic, one can argue that it is okay to use convex hulling in order to find the shape of the edge. The latter is rather expensive. That is why using a bounding capsule is a better alternative.

Since we are only considering roots, this implies that both edges are co-planar (or parallel), (see B.2.3) it is thus allowed to make a 2D projection of the edges.

The direction in which to project is of importance. It is usually the direction that is perpendicular to both edges. If this is non-uniquely defined, another test needs to be done.

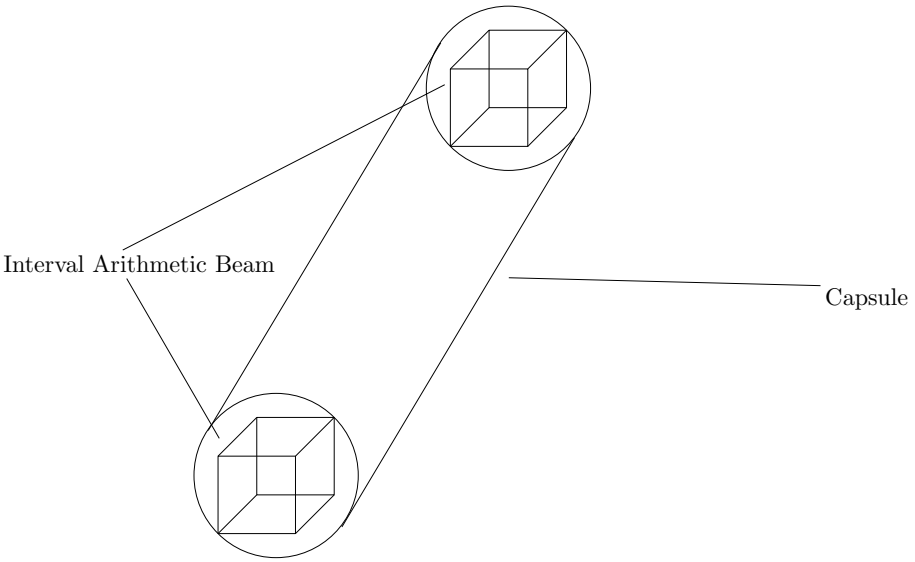

Figure 24: A capsule drawn around two interval beams.

In Figure 25, a simple two-dimensional problem is illustrated. Do the capsules overlap? If one is capable of doing a rectangle/rectangle test, a disk/disk test and a rectangle/disk test, one is capable of answering this question.

As the time domain in which the edge exist gets smaller, both capsules will converge to line segments.

D Interval Arithmetic

An interval is defined by two real numbers $a, b \in \mathbb{R}$, such that $a \leq b$. An interval is closed, meaning that a and b are included within the interval. Let $s = [a, b]$ be an interval. Then s is an element of $\mathbb{IR}$ which is the set of intervals. We can apply unary functions on s. Let $f : \mathbb{R} \mapsto \mathbb{R}$, be a real valued unary function.

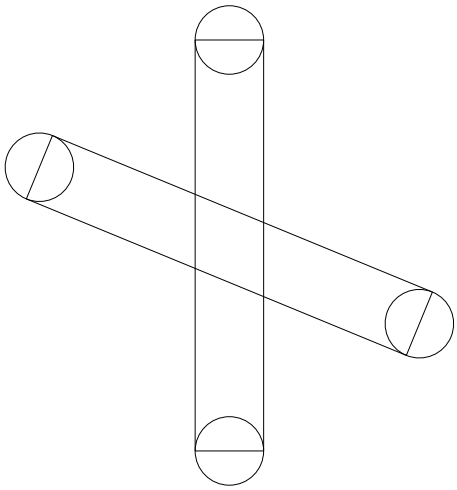

Figure 25: Two capsules, their length and thickness may differ.

An example of such a function is the negation function. But it can also be the sine function.

There exists a concept called a raised function. Instead of mapping elements to elements, such a function maps sets to sets. The same concept can be applied on intervals, but in a different manner. Let $F : \mathbb{IR} \mapsto \mathbb{IR}$ be the "interval raised function" of f.

1. For each $r \in s$ there exists an $r' \in F(s)$ such that $r' = f(r)$.
2. The resulting interval $s' = F(s) = [a, b]$, is minimal. Meaning that if a were any higher or b were any lower, property 1 would not hold.

It is possible to construct a similar definition for binary functions and for functions of arbitrary arity, but this will do little good. Instead, let us define some example operations on intervals. Let $u, v \in \mathbb{IR}$:

$$-u = [-u_{\text{upper}}, -u_{\text{lower}}] \tag{46}$$

$$u + v = [u_{\text{lower}} + v_{\text{lower}}, u_{\text{upper}} + v_{\text{upper}}] \tag{47}$$

$$u - v = u + (-v) \tag{48}$$

$$u \cdot v = \begin{bmatrix} \min\{u_{\text{lower}} \cdot v_{\text{lower}}, u_{\text{lower}} \cdot v_{\text{upper}}, u_{\text{upper}} \cdot v_{\text{lower}}, u_{\text{upper}} \cdot v_{\text{upper}}\}, \\ \max\{u_{\text{lower}} \cdot v_{\text{lower}}, u_{\text{lower}} \cdot v_{\text{upper}}, u_{\text{upper}} \cdot v_{\text{lower}}, u_{\text{upper}} \cdot v_{\text{upper}}\} \end{bmatrix} \tag{49}$$

The operations above are the most common ones. It should be noted that multiplication of two intervals is reasonably in-efficient, due to the max and min operators. It is also possible to construct a division operation. Note that in general $u + v - v \neq u$ and $\frac{u \cdot v}{v} \neq u$. Thus interval arithmetic is not a field.

When creating composed functions, property 2 (property of minimality), might fail. There exist some theorems about the minimality of composed functions, but in general, composed functions do not have a minimal interval representation.

In fact, composed functions are only required to give an upper-bound and a lower-bound. An additional property for many algorithms to function is that the result converges to a single value as the input converges to a single value.

E Taylor models

In [25], the concept of Taylor models as a substitute for interval arithmetic is used. Taylor models are more thoroughly defined in [14]. A Taylor model is a vertically fattened polynomial bound around a more complicated function. In [25] a definition of a Taylor model is given. The same definition is repeated over here (after correcting a tiny mistake):

> Let the Taylor model be defined over an interval, i.e. $t \in [t_0, t_1]$. Let $m \in [t_0, t_1]$ be the point around which the Taylor-polynomial used in the Taylor model is developed. Let $f(t)$ be the function that is included within the Taylor model, then:
>
> $$f(t) \in \sum_{i=0}^{n} \frac{d^i f(m)}{dt^i \cdot i!}(t - m)^i + [r_0, r_1] \tag{50}$$
>
> Where $[r_0, r_1]$ represents a bound on the error term. This bound represents the amount of absolute deviation that can occur within a specified domain.

It is possible to apply arithmetic operations on Taylor models. The two most important ones are addition and multiplication.

F The Lagrange Remainder

Taylor's theorem with the Lagrange remainder can be written as follows:

$$f(x) = \sum_{i=0}^{n} \frac{d^i f(a)}{dx^i} \frac{(x-a)^i}{i!} + \frac{d^{n+1} f(s)}{dx^{n+1}} \frac{(x-a)^{n+1}}{(n+1)!}. \tag{51}$$

Where $n \in \mathbb{N}$, and s is in between a and x. In order for this theorem to be valid, $f(x)$ needs to be $\mathcal{C}_{n+1}$ continuous (in the interval containing a and x). This description is based on the formulation in [1]. Note that the notation in [1] is better. It has not been used here for consistency reasons.

The term $\sum_{i=0}^{n} \frac{d^i f(a)}{dx^i} \frac{(x-a)^i}{i!}$ is a Taylor polynomial without error term. The term $\frac{d^{n+1} f(s)}{dx^{n+1}} \frac{(x-a)^{n+1}}{(n+1)!}$, is an explicit formulation of the error. This error is a function of s. We know that s is in between a and x, but we do not know

the exact value of s. We do have an upper bound and a lower bound for s. Using the Lagrange remainder in combination with interval arithmetic, can yield surprising results (as is shown in Appendix G).

G Taylor-Lagrange based root finder

Let $f : \mathbb{R} \mapsto \mathbb{R}$ be any function that is $\mathcal{C}_2$ continuous. Let $s \in \mathbb{IR}$, be the search domain of the function. The root finder requires two derived functions namely: $\frac{df}{dt} : \mathbb{R} \mapsto \mathbb{R}$ and $\frac{d^2f}{dt^2} : \mathbb{IR} \mapsto \mathbb{IR}$. The latter function deserves some annotations. It is a mapping from an interval (i.e the current search domain) to an interval of second derivatives. In other words, the result contains a lower bound for the second derivative along the interval and an upper bound for the second derivative along the interval.

Now we can construct a second order Taylor interval polynomial that encloses the function f along the domain s. Let g be a mapping from time to distance interval i.e. $g : \mathbb{R} \mapsto \mathbb{IR}$.

$$g(t) = f(s_{\text{lower}}) + \frac{df(s_{\text{lower}})}{dt}(t - s_{\text{lower}}) + \frac{d^2f(s)}{dt^2}\frac{(t - s_{\text{lower}})^2}{2} \tag{52}$$

The first two terms of the polynomial above look like the development of a normal Taylor polynomial around s_{lower}. The third term however is based on an explicit formulation of the remainder called the Lagrange form. By the definition of the Lagrange remainder, the function f is "included" in g along the search domain. Outside of the search domain, f is no longer required to be "in" g.

Whenever a function g is constructed for a given search domain s, the roots of g (there are at most two) can be computed. However, it seems to be easier to compute the roots of the lower bound of the polynomial separately from the roots of the upper-bound of the polynomial (see Figure 26).

When looking at the Figure, one might notice that the upper bounding polynomial does not hit the horizontal axis (yet). In the situation at $t = t_7$, it is still uncertain whether the function will hit the horizontal axis or not. However, as t approaches the root, both curves will construct a fit around f that is tighter.

In the picture, the second derivative interval is based on the entire domain excluding the time between the start of the domain and the "current" time. If both the lower bounding parabola and the upper bounding parabola intersect the horizontal axis, then there is certainty about intersection of f with the horizontal axis. The search domain will be reasonably small then, and will thus result in a tighter bound on the second derivative, causing the root finder to converge extremely fast ($t = t_8$).

Once the span of the root (the root is an interval), reaches below a certain threshold, lets say ϵ_t, then the root interval is returned.

Note that a picture similar to Figure 26 is depicted in [15]. Instead of using upper-bound and lower-bound polynomials, [15] uses only lower-bound polynomials.

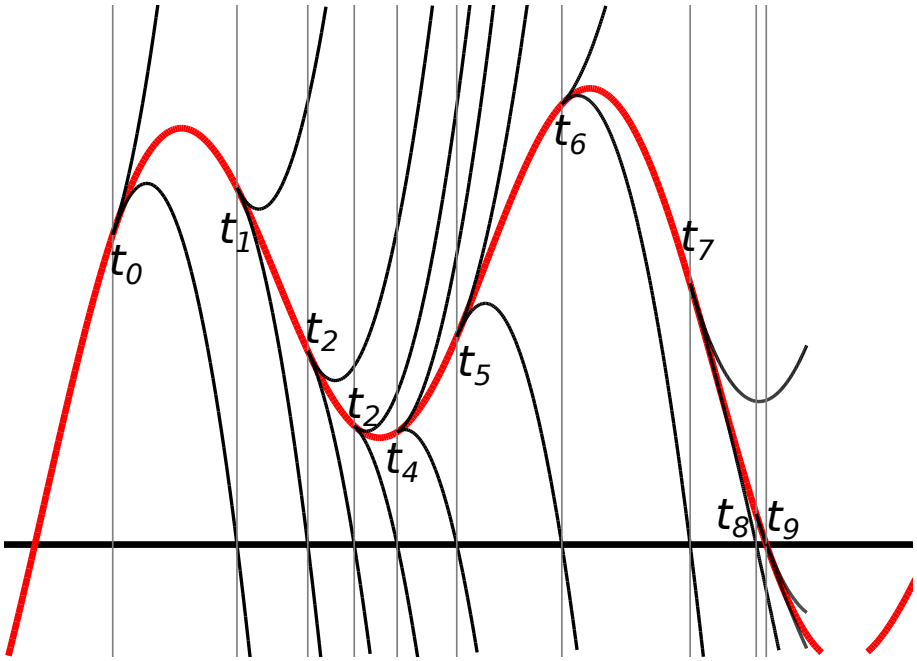

Figure 26: Operation of the Taylor-Lagrange based root finder.

H An example of a 2D Boolean function

```
//! Compute the TOC between an edge and a point

template<typename T>
struct EdgePointTOC : public BinaryFunction<T>
{
        Matrix<T, 3, 3 > aEdgeMatrixInverse; //the y-axis is pointing along the edge
        //Note that the matrix is orthonormal and translated (the inverse also has this property)
        //the origin of the homogeneous matrix is edge0
        Vector<T, 2 > bPoint; //a point in B-space
        T edgeLength; //the length of the edge

        GLMObject<T> aGLM; //the motion of object A
        GLMObject<T> bGLM; //the motion of object B

        bool operator() (const Interval<T> &domain)
        {
                Matrix<Interval<T>, 3, 3 > Ainverse, B, C;

                Ainverse = aEdgeMatrixInverse.template cast<Interval<T> >() * aGLM.getInverseMatrix(domain);
                //a inverse is a transformation from world space to A edge space

                B = bGLM.getMatrix(domain);
                //B is a transformation from B object space to world space

                C = Ainverse * B;
                //C is a transformation from B object space to A edge space

                Vector<Interval<T>, 3 > pointInEdgeSpace = C * bPoint.toHomogeneous().template cast<Interval<T> >();

                Interval<T> edge(0, edgeLength);

                if (pointInEdgeSpace[1].overlaps(edge) && pointInEdgeSpace[0].lower() <= 0 && pointInEdgeSpace[0].upper() >= 0)
                        return true;

                return false;
        }

};
```

I A proof that the parabolas are indeed bounding parabolas

Consider this arbitrary interval function $h : \mathbb{IR} \mapsto \mathbb{IR}$. Let the same function also be defined on real numbers $h : \mathbb{R} \mapsto \mathbb{R}$. Now let $b \in \mathbb{IR}$ and let $a \in \mathbb{R}$ and $a \in b$. Now, by definition, the following inequalities hold:

$$h(b)_{\text{lower}} \leq h(a) \leq h(b)_{\text{upper}}. \tag{53}$$

A similar statement: $h(a) \in h(b)$ is also true.

For any $t \in (s_{\text{lower}}, s_{\text{upper}}]$, there exists a t^*, such that $s_{\text{lower}} < t^* < t$ for which:

$$f(t) = f(s_{\text{lower}}) + \frac{df(s_{\text{lower}})}{dt}(t - s_{\text{lower}}) + \frac{d^2 f(t^*)}{dt^2}\frac{(t - s_{\text{lower}})^2}{2} \tag{54}$$

Note that the above is just a restricted version of Taylor's theorem with the Lagrange remainder.

Note that for any $t \in (s_{\text{lower}}, s_{\text{upper}}]$, the expression $\frac{(t-s_{\text{lower}})^2}{2}$ is just a positive scalar. The function $\frac{d^2 f(t^*)}{dt^2}$ is a mapping from real to real. Raising this function to interval arithmetic and giving it the parameter s would give us the in-equalities:

$$\frac{d^2 f(s)}{dt^2}_{\text{lower}} \leq \frac{d^2 f(t^*)}{dt^2} \leq \frac{d^2 f(s)}{dt^2}_{\text{upper}}. \tag{55}$$

By multiplying with our positive constant, the in-equalities will be conserved:

$$\frac{d^2 f(s)}{dt^2}_{\text{lower}} \frac{(t - s_{\text{lower}})^2}{2} \leq \frac{d^2 f(t^*)}{dt^2}\frac{(t - s_{\text{lower}})^2}{2} \leq \frac{d^2 f(s)}{dt^2}_{\text{upper}} \frac{(t - s_{\text{lower}})^2}{2}. \tag{56}$$

Summarizing what is stated above, for any t within $(s_{\text{lower}}, s_{\text{upper}}]$, the exact error is bounded if we supply the entire domain s to the raised $\frac{d^2 f}{dt^2}$ function.

As an additional remark, it can be noted that the domain in which t must lie is open at s_{lower}. If t were to get the value of s_{lower} then the error term would not be relevant (i.e. 0). In this case the bounding function is exact.

J A proof that the parabolas are indeed diverging

Recall the inclusion equation 52, the term with the largest exponent is an interval, meaning that $\frac{d^2 f(s)}{dt^2}\frac{(t-s_{\text{lower}})^2}{2}$ is an interval. The term $\frac{d^2 f(s)}{dt^2}$ represents the second time derivative over the interval s. Again, we get an upper bound and a lower bound. Because $\frac{(t-s_{\text{lower}})^2}{2}$, is a non negative real expression, that is 0 at s_{lower}, the parabolas are diverging as they get away from s_{lower}.

This is always the case, unless the second interval-derivative span is 0. In this case, the inclusion function is a perfect bound.

K Processed Data

The data below has been constructed using a high level of automation. Although the benchmark conclusions are written in text, they have been automatically generated in order to avoid mistakes. It must be noted that timings are measured in milliseconds (instead of seconds) and that distance is measured in units, though velocities are measured in units per second. Each measurement is recorded by repeating it 10 times. The time recorded is then divided by 10. There is one exception however, the tetrahedron versus tetrahedron experiments have been repeated 1000 times. This is because these timings are rather low with respect to the other experiments. If the tetrahedron versus tetrahedron experiments would have been done only 10 times, this would ruin the worst case relative overall accuracy of the 40 benchmarks.

Because all C2A timings are performed twice, this serves as a sanity check, in order to assert that the timings are correct.

Whenever the word significantly is used within a comparison, this means that the relative magnitude between the two quantities differs by at least a factor 1.5.

In the general remark sections, "facts" are described. For the speed comparisons, these facts are based on the mean timings. For accuracy comparisons, the facts are based on the worst case accuracy of the benchmark.

K.1 Benchmark 0/1 Tetrahedron versus Tetrahedron (low speed)

	Time	Dist.	v	$v \cdot \hat{n}$	TOC
Minimum	0.16	$1.0 \cdot 10^{-10}$	3.0	1.3	0.66
Lower Quartile	0.77	$1.0 \cdot 10^{-10}$	3.9	2.9	0.76
Median	1.2	$1.0 \cdot 10^{-10}$	4.2	3.5	0.79
Upper Quartile	2.3	$1.1 \cdot 10^{-10}$	4.4	3.9	0.81
Maximum	25.	$1.4 \cdot 10^{-10}$	5.3	4.9	0.89
Mean	2.1	$1.0 \cdot 10^{-10}$	4.2	3.4	0.78
Standard-Deviation	2.8	$4.4 \cdot 10^{-12}$	0.41	0.72	0.043

Table 1: D2M statistics

	Time	Dist.	v	$v \cdot \hat{n}$	TOC
Minimum	0.076	0	-	-	0.66
Lower Quartile	0.098	0.00024	-	-	0.76
Median	0.10	0.00032	-	-	0.79
Upper Quartile	0.11	0.00040	-	-	0.81
Maximum	0.14	0.00060	-	-	0.89
Mean	0.11	0.00032	-	-	0.78
Standard-Deviation	0.012	0.00011	-	-	0.043

Table 2: C2A statistics

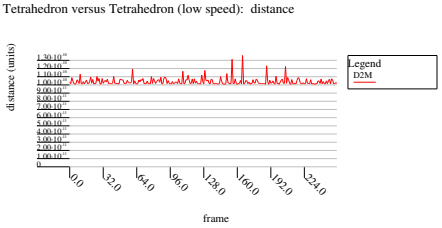

Figure 27: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	0.058	0.0096	4.0	-	0.60
Lower Quartile	0.12	0.040	4.0	-	0.74
Median	0.20	0.050	4.1	-	0.77
Upper Quartile	0.29	0.062	4.1	-	0.80
Maximum	1.2	0.090	4.4	-	0.87
Mean	0.24	0.051	4.1	-	0.77
Standard-Deviation	0.17	0.016	0.080	-	0.043

Table 3: D2M Capoeira statistics

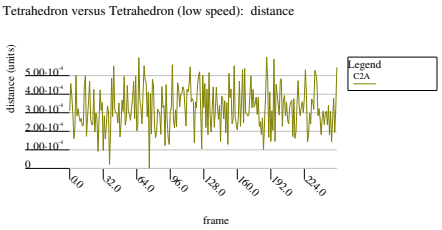

Figure 28: A plot

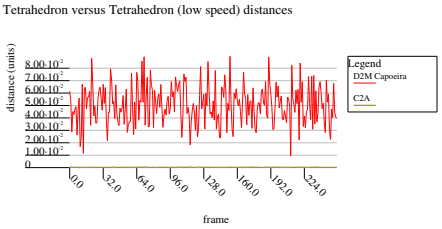

Figure 29: A dual plot

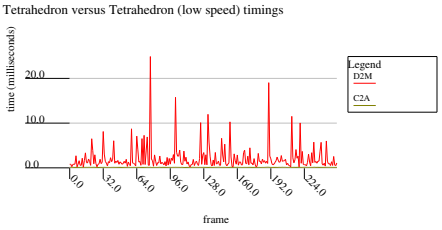

Figure 30: A dual plot

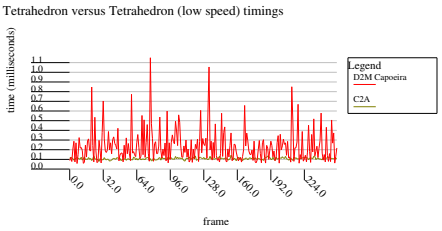

Figure 31: A dual plot

Worst Case Timing Remarks

C2A is $1.7 \cdot 10^2$ times faster than D2M. C2A is 8.1 times faster than D2M Capoeira. D2M Capoeira is 22. times faster than D2M.

Average Timing Remarks

C2A is 20. times faster than D2M. C2A is 2.2 times faster than D2M Capoeira. D2M Capoeira is 8.9 times faster than D2M.

Worst Case Accuracy Remarks

D2M is $4.4 \cdot 10^6$ times more accurate than C2A. C2A is $1.5 \cdot 10^2$ times more accurate than D2M Capoeira. D2M is $6.6 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 76. D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. C2A is significantly faster than D2M Capoeira. C2A is significantly more accurate than D2M Capoeira. The worst case recorded absolute timing error of C2A is 0.027 milliseconds.

K.2 Benchmark 2/3 Tetrahedron versus Tetrahedron (high speed)

	Time	Dist.	v	v · n̂	TOC
Minimum	0.31	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$1.7 \cdot 10^{2}$	1.0
Lower Quartile	0.83	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.7 \cdot 10^{2}$	1.0
Median	1.4	$1.5 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.3 \cdot 10^{2}$	1.0
Upper Quartile	2.8	$1.7 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.8 \cdot 10^{2}$	1.0
Maximum	15.	$3.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	2.3	$1.6 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.2 \cdot 10^{2}$	1.0
Standard-Deviation	2.2	$3.0 \cdot 10^{-11}$	0.40	73.	0.00034

Table 4: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.072	$3.2 \cdot 10^{-5}$	-	-	1.0
Lower Quartile	0.080	0.00056	-	-	1.0
Median	0.084	0.0024	-	-	1.0
Upper Quartile	0.090	0.0081	-	-	1.0
Maximum	0.11	0.038	-	-	1.0
Mean	0.085	0.0071	-	-	1.0
Standard-Deviation	0.0067	0.0099	-	-	0.00035

Table 5: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.062	0.016	$5.1 \cdot 10^{2}$	-	1.0
Lower Quartile	0.13	0.038	$5.1 \cdot 10^{2}$	-	1.0
Median	0.20	0.049	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	0.31	0.060	$5.1 \cdot 10^{2}$	-	1.0
Maximum	1.1	0.095	$5.1 \cdot 10^{2}$	-	1.0
Mean	0.25	0.050	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.17	0.016	0.00067	-	0.00034

Table 6: D2M Capoeira statistics

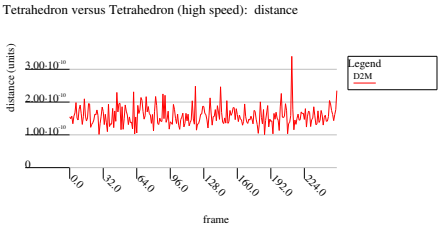

Figure 32: A plot

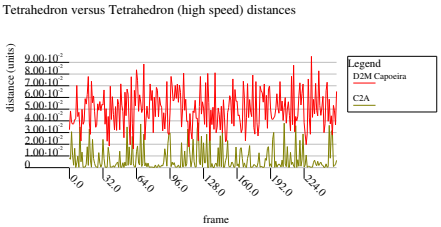

Figure 34: A dual plot

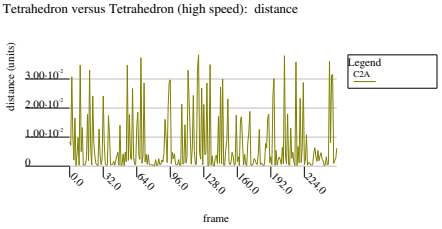

Figure 33: A plot

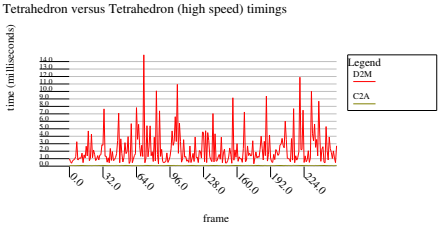

Figure 35: A dual plot

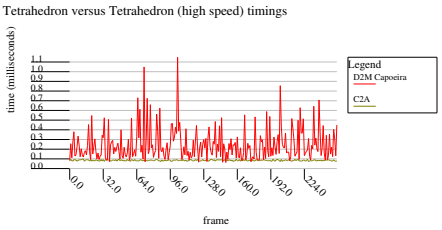

Figure 36: A dual plot

Worst Case Timing Remarks

C2A is $1.4 \cdot 10^2$ times faster than D2M. C2A is 11. times faster than D2M Capoeira. D2M Capoeira is 13. times faster than D2M.

Average Timing Remarks

C2A is 27. times faster than D2M. C2A is 2.9 times faster than D2M Capoeira. D2M Capoeira is 9.1 times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.1 \cdot 10^8$ times more accurate than C2A. C2A is 2.5 times more accurate than D2M Capoeira. D2M is $2.8 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. C2A is significantly faster than D2M Capoeira. C2A is significantly more accurate than D2M Capoeira. The worst case recorded absolute timing error of C2A is 0.025 milliseconds.

K.3 Benchmark 4/5 Tetrahedron versus Bunny (low speed)

	Time	Dist.	v	$v \cdot \hat{n}$	TOC
Minimum	1.9	$1.0 \cdot 10^{-10}$	2.5	0.50	0.54
Lower Quartile	13.	$1.0 \cdot 10^{-10}$	4.0	3.1	0.69
Median	28.	$1.0 \cdot 10^{-10}$	4.3	3.6	0.73
Upper Quartile	44.	$1.1 \cdot 10^{-10}$	4.5	4.0	0.76
Maximum	$1.9 \cdot 10^2$	$1.3 \cdot 10^{-10}$	5.7	5.1	0.86
Mean	35.	$1.0 \cdot 10^{-10}$	4.2	3.5	0.73
Standard-Deviation	30.	$3.4 \cdot 10^{-12}$	0.43	0.78	0.050

Table 7: D2M statistics

	Time	Dist.	v	$v \cdot \hat{n}$	TOC
Minimum	0.19	0	-	-	0.54
Lower Quartile	3.7	0.00022	-	-	0.69
Median	17.	0.00032	-	-	0.73
Upper Quartile	30.	0.00041	-	-	0.76
Maximum	$1.6 \cdot 10^2$	0.00058	-	-	0.86
Mean	22.	0.00031	-	-	0.73
Standard-Deviation	24.	0.00013	-	-	0.050

Table 8: C2A statistics

	Time	Dist.	v	$v \cdot \hat{n}$	TOC
Minimum	0.084	0.0056	4.0	-	0.53
Lower Quartile	0.40	0.026	4.0	-	0.68
Median	0.68	0.036	4.1	-	0.72
Upper Quartile	1.1	0.048	4.2	-	0.75
Maximum	2.6	0.064	4.7	-	0.84
Mean	0.79	0.036	4.1	-	0.72
Standard-Deviation	0.51	0.013	0.12	-	0.049

Table 9: D2M Capoeira statistics

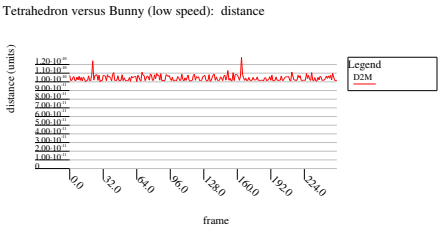

Figure 37: A plot

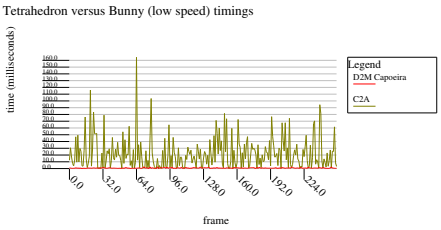

Figure 41: A dual plot

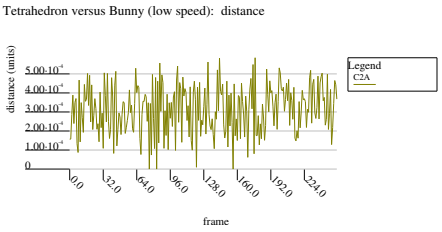

Figure 38: A plot

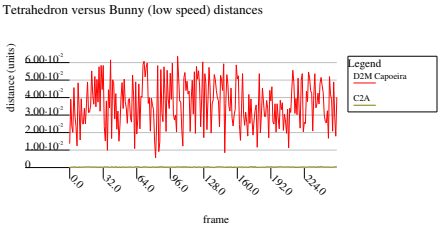

Figure 39: A dual plot

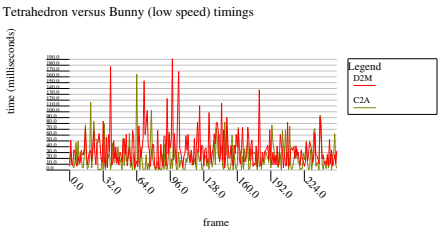

Figure 40: A dual plot

Worst Case Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 62. times faster than C2A. D2M Capoeira is 72. times faster than D2M.

Average Timing Remarks

C2A is 1.6 times faster than D2M. D2M Capoeira is 28. times faster than C2A. D2M Capoeira is 44. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $4.6 \cdot 10^6$ times more accurate than C2A. C2A is $1.1 \cdot 10^2$ times more accurate than D2M Capoeira. D2M is $5.0 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 76, 83 and 156. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 1.3 milliseconds.

K.4 Benchmark 6/7 Tetrahedron versus Bunny (high speed)

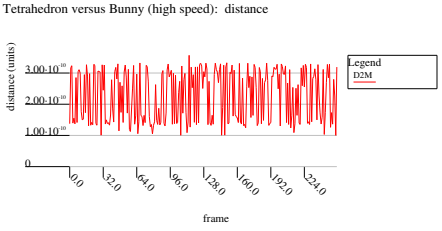

Figure 42: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	5.1	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	66.	1.0
Lower Quartile	13.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.9 \cdot 10^{2}$	1.0
Median	21.	$2.5 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.5 \cdot 10^{2}$	1.0
Upper Quartile	44.	$3.1 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.8 \cdot 10^{2}$	1.0
Maximum	$2.0 \cdot 10^{2}$	$3.6 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	34.	$2.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.3 \cdot 10^{2}$	1.0
Standard-Deviation	33.	$8.2 \cdot 10^{-11}$	0.41	79.	0.00038

Table 10: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	$9.1 \cdot 10^{-5}$	-	-	1.0
Lower Quartile	2.0	0.0061	-	-	1.0
Median	7.0	0.015	-	-	1.0
Upper Quartile	13.	0.027	-	-	1.0
Maximum	41.	0.050	-	-	1.0
Mean	8.9	0.018	-	-	1.0
Standard-Deviation	8.5	0.014	-	-	0.00039

Table 11: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	0.0094	$5.1 \cdot 10^{2}$	-	1.0
Lower Quartile	0.43	0.027	$5.1 \cdot 10^{2}$	-	1.0
Median	0.66	0.036	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	1.0	0.045	$5.1 \cdot 10^{2}$	-	1.0
Maximum	2.8	0.070	$5.1 \cdot 10^{2}$	-	1.0
Mean	0.78	0.036	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.49	0.013	0.00097	-	0.00038

Table 12: D2M Capoeira statistics

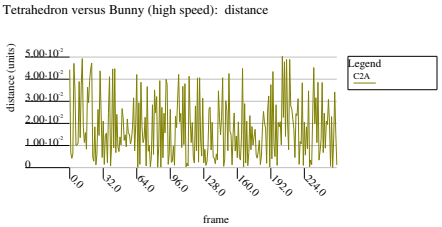

Figure 43: A plot

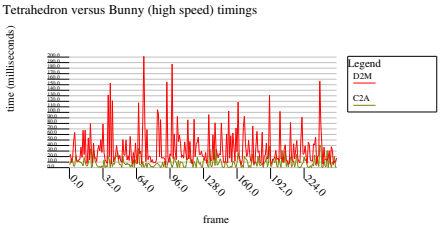

Figure 45: A dual plot

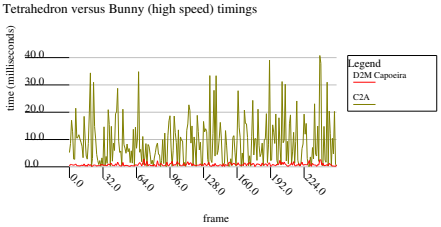

Figure 46: A dual plot

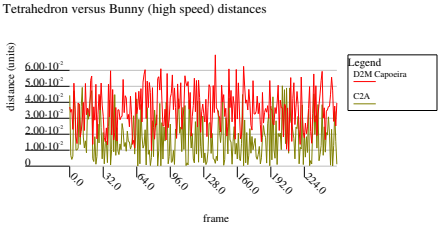

Figure 44: A dual plot

Worst Case Timing Remarks

C2A is 4.9 times faster than D2M. D2M Capoeira is 15. times faster than

C2A. D2M Capoeira is 71. times faster than D2M.

Average Timing Remarks

C2A is 3.8 times faster than D2M. D2M Capoeira is 11. times faster than C2A. D2M Capoeira is 43. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.4 \cdot 10^{8}$ times more accurate than C2A. C2A is not significantly more accurate than D2M Capoeira, nor is it less accurate than D2M Capoeira. D2M is $2.0 \cdot 10^{8}$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.59 milliseconds.

K.5 Benchmark 8/9 Tetrahedron versus Dragon (low speed)

	Time	Dist.	v	$\mathbf{v} \cdot \hat{\mathbf{n}}$	TOC
Minimum	1.2	$1.0 \cdot 10^{-10}$	2.8	0.47	0.42
Lower Quartile	5.7	$1.0 \cdot 10^{-10}$	4.1	2.9	0.55
Median	9.9	$1.0 \cdot 10^{-10}$	4.5	3.6	0.62
Upper Quartile	16.	$1.1 \cdot 10^{-10}$	4.8	4.2	0.69
Maximum	$2.6 \cdot 10^{2}$	$1.2 \cdot 10^{-10}$	6.3	6.0	0.86
Mean	14.	$1.0 \cdot 10^{-10}$	4.5	3.5	0.63
Standard-Deviation	19.	$3.7 \cdot 10^{-12}$	0.59	1.0	0.10

Table 13: D2M statistics

	Time	Dist.	v	$\mathbf{v} \cdot \hat{\mathbf{n}}$	TOC
Minimum	0.26	0	-	-	0.42
Lower Quartile	5.9	0.00026	-	-	0.55
Median	16.	0.00035	-	-	0.62
Upper Quartile	37.	0.00047	-	-	0.69
Maximum	$2.1 \cdot 10^{2}$	0.00074	-	-	0.86
Mean	25.	0.00036	-	-	0.63
Standard-Deviation	28.	0.00015	-	-	0.10

Table 14: C2A statistics

	Time	Dist.	v	$\mathbf{v} \cdot \hat{\mathbf{n}}$	TOC
Minimum	0.10	0.013	4.0	-	0.41
Lower Quartile	0.48	0.032	4.1	-	0.54
Median	0.85	0.042	4.1	-	0.60
Upper Quartile	1.3	0.050	4.4	-	0.67
Maximum	4.5	0.14	5.1	-	0.85
Mean	0.95	0.042	4.2	-	0.61
Standard-Deviation	0.64	0.014	0.24	-	0.099

Table 15: D2M Capoeira statistics

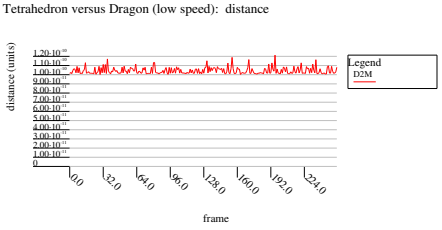

Figure 47: A plot

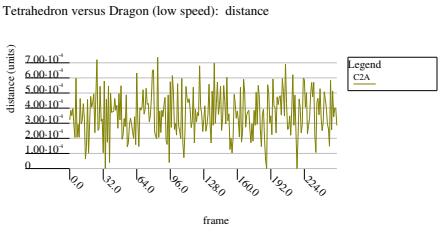

Figure 48: A plot

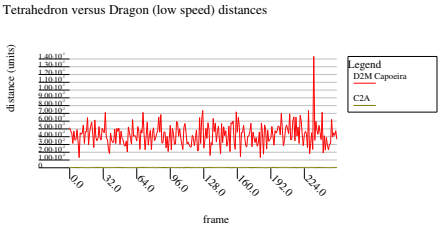

Figure 49: A dual plot

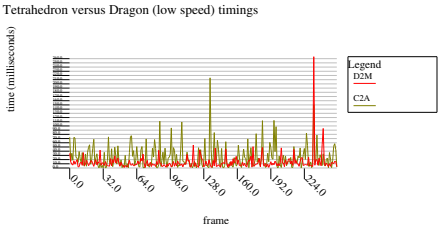

Figure 50: A dual plot

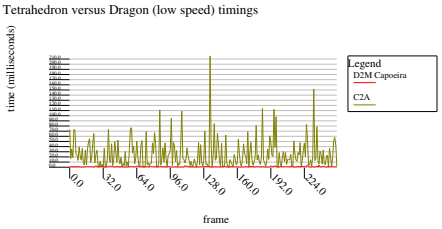

Figure 51: A dual plot

Worst Case Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 48. times faster than C2A. D2M Capoeira is 59. times faster than D2M.

Average Timing Remarks

D2M is 1.8 times faster than C2A. D2M Capoeira is 27. times faster than C2A. D2M Capoeira is 14. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $6.1 \cdot 10^{6}$ times more accurate than C2A. C2A is $1.9 \cdot 10^{2}$ times more accurate than D2M Capoeira. D2M is $1.2 \cdot 10^{9}$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 34, 188 and 217. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 2.9 milliseconds.

K.6 Benchmark 10/11 Tetrahedron versus Dragon (high speed)

	Time	Dist.	v	v · n̂	TOC
Minimum	1.6	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	81.	1.0
Lower Quartile	5.2	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.3 \cdot 10^{2}$	1.0
Median	8.3	$1.8 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.1 \cdot 10^{2}$	1.0
Upper Quartile	16.	$2.6 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.7 \cdot 10^{2}$	1.0
Maximum	70.	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	12.	$2.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.9 \cdot 10^{2}$	1.0
Standard-Deviation	11.	$7.1 \cdot 10^{-11}$	0.56	99.	0.00085

Table 16: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.17	0.00014	-	-	0.99
Lower Quartile	1.6	0.0098	-	-	1.0
Median	4.5	0.021	-	-	1.0
Upper Quartile	11.	0.032	-	-	1.0
Maximum	58.	0.050	-	-	1.0
Mean	7.9	0.021	-	-	1.0
Standard-Deviation	9.2	0.014	-	-	0.00084

Table 17: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	0.013	$5.1 \cdot 10^2$	-	1.0
Lower Quartile	0.42	0.029	$5.1 \cdot 10^2$	-	1.0
Median	0.68	0.038	$5.1 \cdot 10^2$	-	1.0
Upper Quartile	1.0	0.048	$5.1 \cdot 10^2$	-	1.0
Maximum	3.0	0.075	$5.1 \cdot 10^2$	-	1.0
Mean	0.79	0.040	$5.1 \cdot 10^2$	-	1.0
Standard-Deviation	0.49	0.013	0.0022	-	0.00085

Table 18: D2M Capoeira statistics

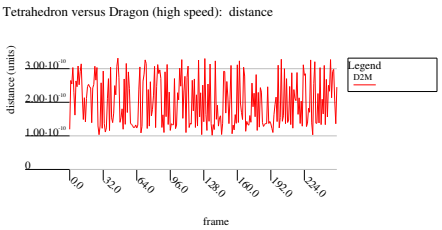

Figure 52: A plot

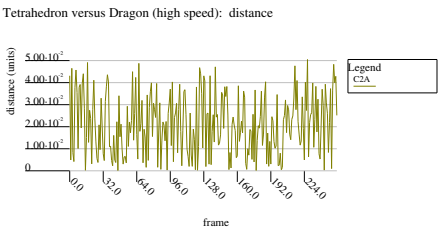

Figure 53: A plot

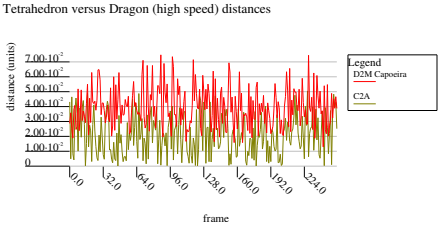

Figure 54: A dual plot

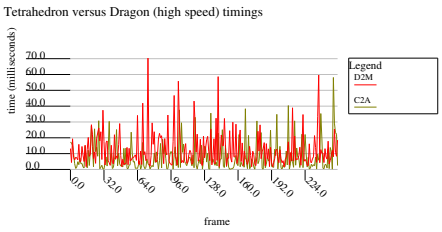

Figure 55: A dual plot

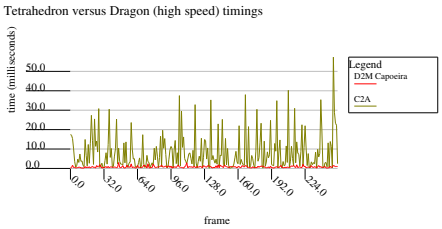

Figure 56: A dual plot

Worst Case Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 19. times faster than C2A. D2M Capoeira is 23. times faster than D2M.

Average Timing Remarks

C2A is 1.5 times faster than D2M. D2M Capoeira is 10. times faster than

C2A. D2M Capoeira is 16. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^{8}$ times more accurate than C2A. C2A is not significantly more accurate than D2M Capoeira, nor is it less accurate than D2M Capoeira. D2M is $2.3 \cdot 10^{8}$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.79 milliseconds.

K.7 Benchmark 12/13 Tetrahedron versus Buddha (low speed)

	Time	Dist.	v	v · n̂	TOC
Minimum	0.99	$1.0 \cdot 10^{-10}$	2.6	0.80	0.52
Lower Quartile	8.0	$1.0 \cdot 10^{-10}$	4.0	2.8	0.67
Median	15.	$1.0 \cdot 10^{-10}$	4.2	3.4	0.74
Upper Quartile	24.	$1.1 \cdot 10^{-10}$	4.5	3.9	0.79
Maximum	$1.5 \cdot 10^{2}$	$1.3 \cdot 10^{-10}$	5.5	5.3	0.87
Mean	19.	$1.0 \cdot 10^{-10}$	4.2	3.3	0.73
Standard-Deviation	17.	$4.9 \cdot 10^{-12}$	0.49	0.86	0.077

Table 19: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.34	$4.2 \cdot 10^{-5}$	-	-	0.52
Lower Quartile	3.8	0.00024	-	-	0.67
Median	18.	0.00033	-	-	0.74
Upper Quartile	49.	0.00041	-	-	0.79
Maximum	$3.3 \cdot 10^{2}$	0.00071	-	-	0.87
Mean	37.	0.00033	-	-	0.73
Standard-Deviation	50.	0.00012	-	-	0.077

Table 20: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.10	0.0086	4.0	-	0.49
Lower Quartile	0.27	0.031	4.0	-	0.66
Median	0.46	0.039	4.1	-	0.73
Upper Quartile	0.67	0.049	4.2	-	0.78
Maximum	1.9	0.089	4.6	-	0.84
Mean	0.51	0.040	4.1	-	0.71
Standard-Deviation	0.32	0.014	0.14	-	0.077

Table 21: D2M Capoeira statistics

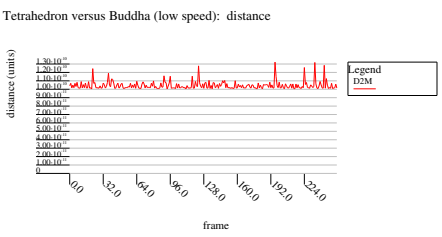

Figure 57: A plot

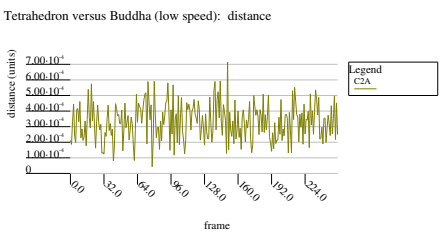

Figure 58: A plot

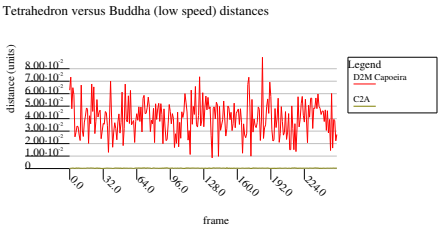

Figure 59: A dual plot

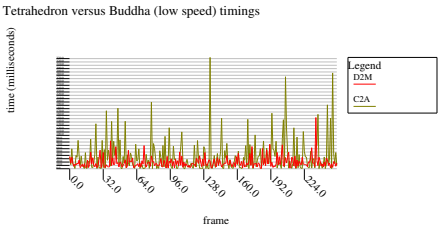

Figure 60: A dual plot

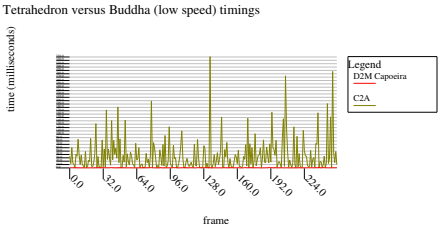

Figure 61: A dual plot

Worst Case Timing Remarks

D2M is 2.2 times faster than C2A. D2M Capoeira is $1.8 \cdot 10^2$ times faster than C2A. D2M Capoeira is 83. times faster than D2M.

Average Timing Remarks

D2M is 1.9 times faster than C2A. D2M Capoeira is 72. times faster than C2A. D2M Capoeira is 37. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $5.4 \cdot 10^6$ times more accurate than C2A. C2A is $1.2 \cdot 10^2$ times more accurate than D2M Capoeira. D2M is $6.7 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly faster than C2A. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 3.0 milliseconds.

K.8 Benchmark 14/15 Tetrahedron versus Buddha (high speed)

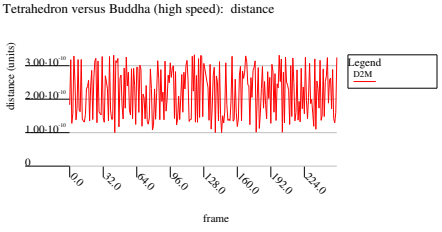

Figure 62: A plot

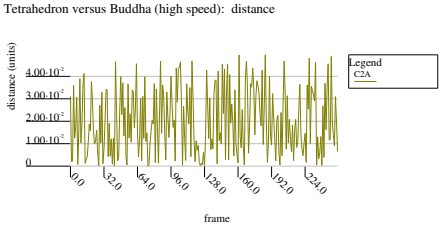

Figure 63: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	1.2	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$1.1 \cdot 10^{2}$	1.0
Lower Quartile	7.5	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.4 \cdot 10^{2}$	1.0
Median	12.	$2.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.1 \cdot 10^{2}$	1.0
Upper Quartile	21.	$2.9 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.7 \cdot 10^{2}$	1.0
Maximum	$1.2 \cdot 10^{2}$	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	18.	$2.2 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.0 \cdot 10^{2}$	1.0
Standard-Deviation	17.	$7.3 \cdot 10^{-11}$	0.47	87.	0.00061

Table 22: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	0	-	-	1.0
Lower Quartile	1.4	0.0081	-	-	1.0
Median	7.7	0.017	-	-	1.0
Upper Quartile	18.	0.032	-	-	1.0
Maximum	72.	0.050	-	-	1.0
Mean	13.	0.020	-	-	1.0
Standard-Deviation	14.	0.014	-	-	0.00062

Table 23: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.11	0.0076	$5.1 \cdot 10^{2}$	-	1.0
Lower Quartile	0.27	0.029	$5.1 \cdot 10^{2}$	-	1.0
Median	0.43	0.037	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	0.72	0.049	$5.1 \cdot 10^{2}$	-	1.0
Maximum	1.8	0.084	$5.1 \cdot 10^{2}$	-	1.0
Mean	0.52	0.038	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.33	0.014	0.0012	-	0.00061

Table 24: D2M Capoeira statistics

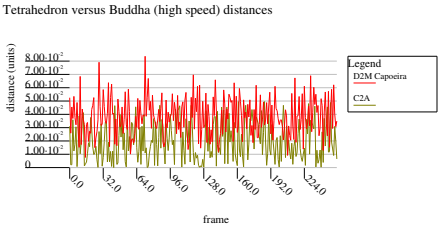

Figure 64: A dual plot

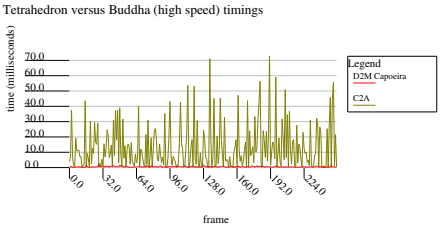

Figure 66: A dual plot

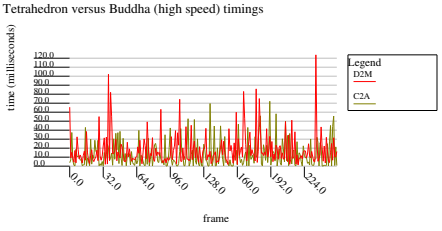

Figure 65: A dual plot

Worst Case Timing Remarks

C2A is 1.7 times faster than D2M. D2M Capoeira is 40. times faster than C2A. D2M Capoeira is 68. times faster than D2M.

Average Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 25. times faster than C2A. D2M Capoeira is 34. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. C2A is 1.7 times more accurate than D2M Capoeira. D2M is $2.5 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 42. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 1.5 milliseconds.

K.9 Benchmark 16/17 Bunny versus Bunny (low speed)

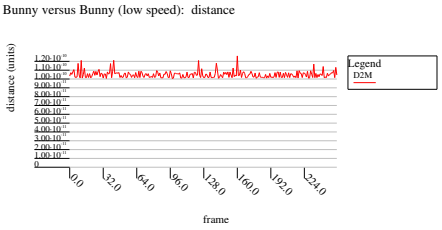

Figure 67: A plot

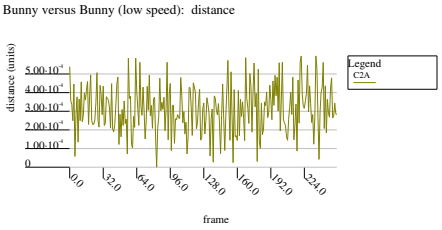

Figure 68: A plot

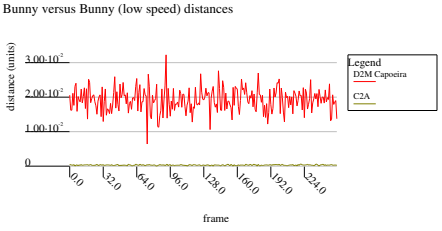

Figure 69: A dual plot

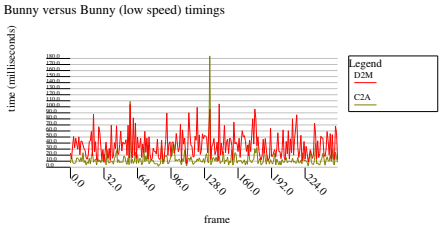

Figure 70: A dual plot

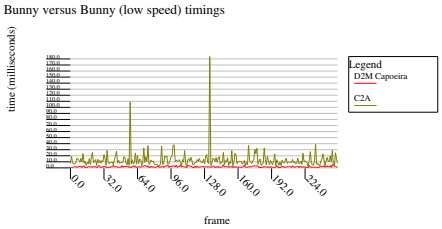

Figure 71: A dual plot

	Time	Dist.	v	v · n̂	TOC
Minimum	5.6	$1.0 \cdot 10^{-10}$	2.6	1.1	0.50
Lower Quartile	20.	$1.0 \cdot 10^{-10}$	4.0	3.2	0.63
Median	32.	$1.0 \cdot 10^{-10}$	4.3	3.7	0.68
Upper Quartile	48.	$1.1 \cdot 10^{-10}$	4.5	4.0	0.71
Maximum	$1.0 \cdot 10^{2}$	$1.3 \cdot 10^{-10}$	5.7	5.6	0.78
Mean	36.	$1.1 \cdot 10^{-10}$	4.3	3.6	0.67
Standard-Deviation	21.	$3.9 \cdot 10^{-12}$	0.42	0.73	0.055

Table 25: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	1.5	0	-	-	0.50
Lower Quartile	7.1	0.00023	-	-	0.63
Median	10.	0.00031	-	-	0.68
Upper Quartile	15.	0.00040	-	-	0.71
Maximum	$1.8 \cdot 10^{2}$	0.00060	-	-	0.78
Mean	13.	0.00031	-	-	0.67
Standard-Deviation	14.	0.00013	-	-	0.055

Table 26: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.26	0.0064	4.0	-	0.49
Lower Quartile	0.98	0.017	4.0	-	0.62
Median	1.6	0.019	4.1	-	0.67
Upper Quartile	2.5	0.021	4.3	-	0.71
Maximum	5.5	0.032	4.7	-	0.77
Mean	1.8	0.019	4.2	-	0.66
Standard-Deviation	1.0	0.0034	0.15	-	0.054

Table 27: D2M Capoeira statistics

Worst Case Timing Remarks

D2M is 1.8 times faster than C2A. D2M Capoeira is 34. times faster than C2A. D2M Capoeira is 19. times faster than D2M.

Average Timing Remarks

C2A is 2.8 times faster than D2M. D2M Capoeira is 7.2 times faster than C2A. D2M Capoeira is 20. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $4.7 \cdot 10^{6}$ times more accurate than C2A. C2A is 54. times more accurate than D2M Capoeira. D2M is $2.6 \cdot 10^{8}$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 83. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.97 milliseconds.

K.10 Benchmark 18/19 Bunny versus Bunny (high speed)

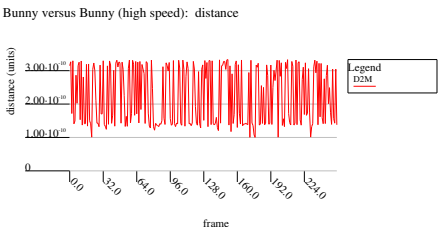

Figure 72: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	9.7	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	63.	1.0
Lower Quartile	26.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.3 \cdot 10^{2}$	1.0
Median	40.	$1.7 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.7 \cdot 10^{2}$	1.0
Upper Quartile	65.	$3.2 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.0 \cdot 10^{2}$	1.0
Maximum	$1.6 \cdot 10^{2}$	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	47.	$2.2 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.5 \cdot 10^{2}$	1.0
Standard-Deviation	27.	$8.6 \cdot 10^{-11}$	0.38	76.	0.00042

Table 28: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.69	0.00044	-	-	1.0
Lower Quartile	2.2	0.0047	-	-	1.0
Median	3.3	0.012	-	-	1.0
Upper Quartile	4.2	0.024	-	-	1.0
Maximum	11.	0.050	-	-	1.0
Mean	3.5	0.016	-	-	1.0
Standard-Deviation	1.7	0.014	-	-	0.00042

Table 29: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.30	0.0098	$5.1 \cdot 10^{2}$	-	1.0
Lower Quartile	1.0	0.018	$5.1 \cdot 10^{2}$	-	1.0
Median	1.6	0.020	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	2.4	0.022	$5.1 \cdot 10^{2}$	-	1.0
Maximum	5.1	0.044	$5.1 \cdot 10^{2}$	-	1.0
Mean	1.8	0.020	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.91	0.0036	0.0013	-	0.00042

Table 30: D2M Capoeira statistics

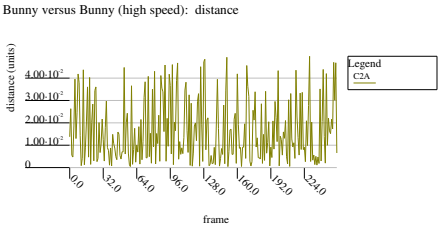

Figure 73: A plot

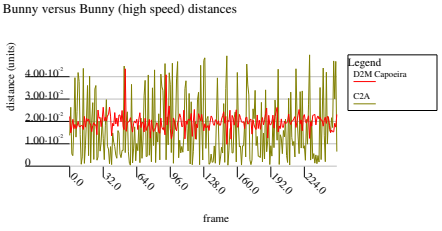

Figure 74: A dual plot

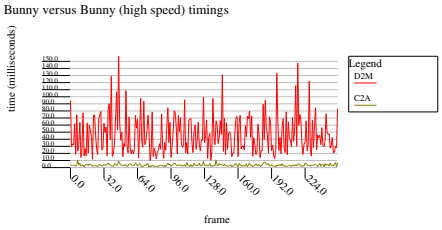

Figure 75: A dual plot

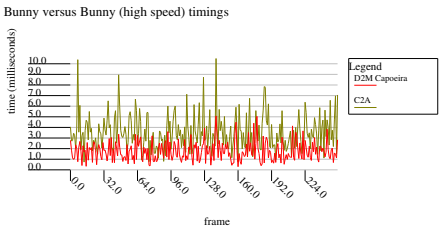

Figure 76: A dual plot

Worst Case Timing Remarks

C2A is 15. times faster than D2M. D2M Capoeira is 2.1 times faster than

C2A. D2M Capoeira is 31. times faster than D2M.

Average Timing Remarks

C2A is 13. times faster than D2M. D2M Capoeira is 2.0 times faster than C2A. D2M Capoeira is 27. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. D2M Capoeira is not significantly more accurate than C2A, nor is it less accurate than C2A. D2M is $1.3 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.49 milliseconds.

K.11 Benchmark 20/21 Bunny versus Dragon (low speed)

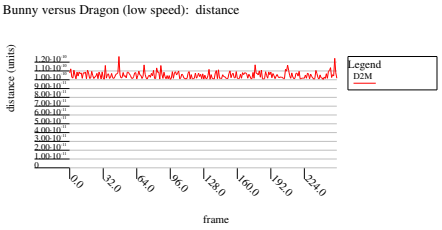

Figure 77: A plot

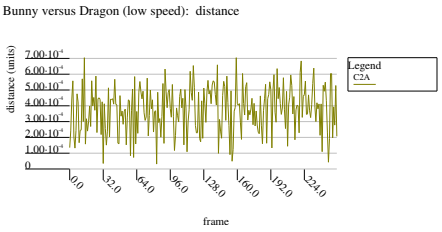

Figure 78: A plot

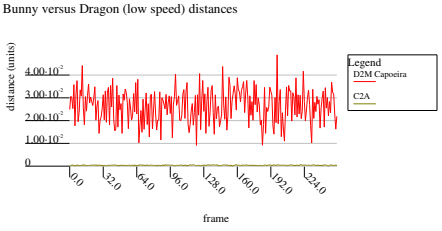

Figure 79: A dual plot

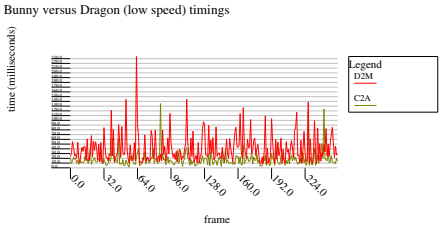

Figure 80: A dual plot

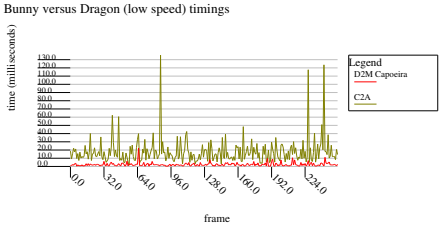

Figure 81: A dual plot

	Time	Dist.	v	v · n̂	TOC
Minimum	7.8	$1.0 \cdot 10^{-10}$	1.7	0.87	0.29
Lower Quartile	20.	$1.0 \cdot 10^{-10}$	4.2	2.8	0.50
Median	30.	$1.0 \cdot 10^{-10}$	4.5	3.6	0.55
Upper Quartile	53.	$1.1 \cdot 10^{-10}$	5.0	4.1	0.62
Maximum	$2.3 \cdot 10^{2}$	$1.3 \cdot 10^{-10}$	6.3	6.1	0.79
Mean	41.	$1.1 \cdot 10^{-10}$	4.5	3.4	0.56
Standard-Deviation	30.	$4.0 \cdot 10^{-12}$	0.65	0.97	0.098

Table 31: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	1.4	$3.1 \cdot 10^{-5}$	-	-	0.29
Lower Quartile	9.1	0.00028	-	-	0.50
Median	15.	0.00039	-	-	0.55
Upper Quartile	21.	0.00049	-	-	0.62
Maximum	$1.4 \cdot 10^{2}$	0.00070	-	-	0.79
Mean	18.	0.00038	-	-	0.56
Standard-Deviation	15.	0.00014	-	-	0.098

Table 32: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.21	0.0091	4.0	-	0.28
Lower Quartile	0.96	0.021	4.1	-	0.49
Median	1.8	0.027	4.2	-	0.54
Upper Quartile	3.0	0.032	4.5	-	0.61
Maximum	22.	0.049	5.8	-	0.78
Mean	2.4	0.026	4.3	-	0.55
Standard-Deviation	2.2	0.0074	0.31	-	0.097

Table 33: D2M Capoeira statistics

Worst Case Timing Remarks

C2A is 1.7 times faster than D2M. D2M Capoeira is 6.0 times faster than C2A. D2M Capoeira is 10. times faster than D2M.

Average Timing Remarks

C2A is 2.3 times faster than D2M. D2M Capoeira is 7.5 times faster than C2A. D2M Capoeira is 17. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $5.6 \cdot 10^{6}$ times more accurate than C2A. C2A is 69. times more accurate than D2M Capoeira. D2M is $3.9 \cdot 10^{8}$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.50 milliseconds.

K.12 Benchmark 22/23 Bunny versus Dragon (high speed)

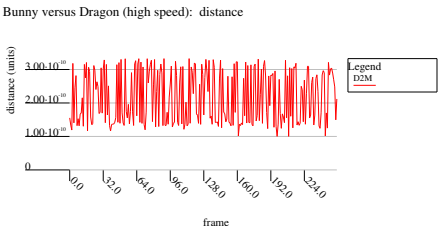

Figure 82: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	7.7	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	70.	0.99
Lower Quartile	22.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.8 \cdot 10^{2}$	1.0
Median	33.	$1.7 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.3 \cdot 10^{2}$	1.0
Upper Quartile	49.	$3.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.8 \cdot 10^{2}$	1.0
Maximum	$1.4 \cdot 10^{2}$	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	39.	$2.2 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.1 \cdot 10^{2}$	1.0
Standard-Deviation	24.	$7.8 \cdot 10^{-11}$	0.60	86.	0.00084

Table 34: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.69	0.0010	-	-	0.99
Lower Quartile	2.6	0.010	-	-	1.0
Median	3.8	0.018	-	-	1.0
Upper Quartile	5.5	0.031	-	-	1.0
Maximum	14.	0.049	-	-	1.0
Mean	4.2	0.020	-	-	1.0
Standard-Deviation	2.3	0.013	-	-	0.00084

Table 35: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.18	0.0089	$5.1 \cdot 10^{2}$	-	0.99
Lower Quartile	0.82	0.020	$5.1 \cdot 10^{2}$	-	1.0
Median	1.3	0.027	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	2.0	0.032	$5.1 \cdot 10^{2}$	-	1.0
Maximum	5.8	0.047	$5.1 \cdot 10^{2}$	-	1.0
Mean	1.5	0.026	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.92	0.0080	0.0028	-	0.00084

Table 36: D2M Capoeira statistics

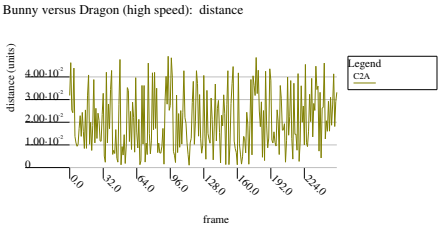

Figure 83: A plot

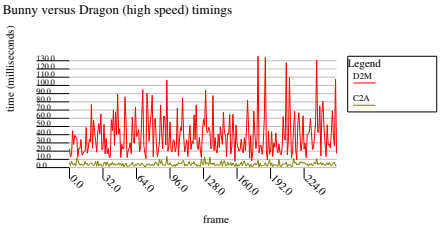

Figure 85: A dual plot

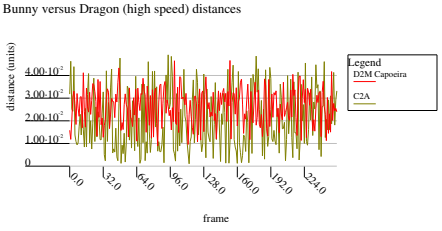

Figure 84: A dual plot

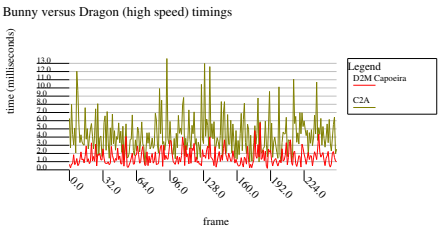

Figure 86: A dual plot

Worst Case Timing Remarks

C2A is 10. times faster than D2M. D2M Capoeira is 2.3 times faster than

C2A. D2M Capoeira is 24. times faster than D2M.

Average Timing Remarks

C2A is 9.2 times faster than D2M. D2M Capoeira is 2.8 times faster than C2A. D2M Capoeira is 25. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. D2M Capoeira is not significantly more accurate than C2A, nor is it less accurate than C2A. D2M is $1.4 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 1.1 milliseconds.

K.13 Benchmark 24/25 Bunny versus Buddha (low speed)

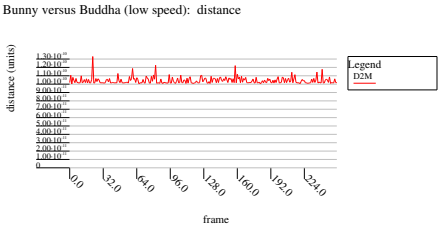

Figure 87: A plot

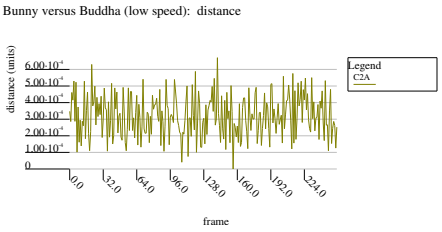

Figure 88: A plot

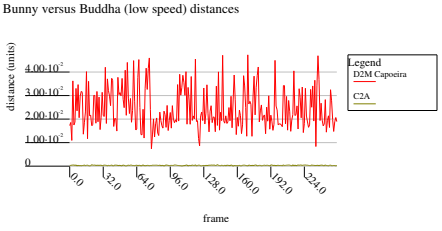

Figure 89: A dual plot

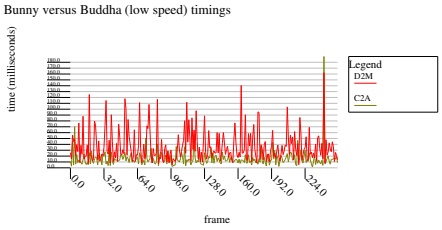

Figure 90: A dual plot

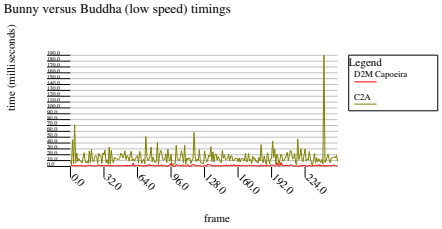

Figure 91: A dual plot

	Time	Dist.	v	v · n̂	TOC
Minimum	5.0	$1.0 \cdot 10^{-10}$	2.7	0.47	0.45
Lower Quartile	18.	$1.0 \cdot 10^{-10}$	4.0	3.2	0.60
Median	28.	$1.0 \cdot 10^{-10}$	4.3	3.6	0.68
Upper Quartile	46.	$1.1 \cdot 10^{-10}$	4.7	4.0	0.72
Maximum	$1.6 \cdot 10^{2}$	$1.3 \cdot 10^{-10}$	5.9	5.6	0.80
Mean	37.	$1.0 \cdot 10^{-10}$	4.3	3.5	0.66
Standard-Deviation	27.	$4.0 \cdot 10^{-12}$	0.52	0.82	0.078

Table 37: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	1.4	0	-	-	0.45
Lower Quartile	8.7	0.00023	-	-	0.60
Median	13.	0.00031	-	-	0.68
Upper Quartile	19.	0.00041	-	-	0.72
Maximum	$1.9 \cdot 10^{2}$	0.00067	-	-	0.80
Mean	16.	0.00032	-	-	0.66
Standard-Deviation	14.	0.00013	-	-	0.078

Table 38: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.11	0.0074	4.0	-	0.43
Lower Quartile	0.85	0.018	4.0	-	0.59
Median	1.2	0.022	4.1	-	0.67
Upper Quartile	1.9	0.030	4.3	-	0.71
Maximum	6.9	0.047	5.0	-	0.79
Mean	1.4	0.025	4.2	-	0.66
Standard-Deviation	0.94	0.0088	0.18	-	0.078

Table 39: D2M Capoeira statistics

Worst Case Timing Remarks

D2M is not significantly faster than C2A nor is it slower than C2A. D2M Capoeira is 28. times faster than C2A. D2M Capoeira is 24. times faster than D2M.

Average Timing Remarks

C2A is 2.4 times faster than D2M. D2M Capoeira is 11. times faster than C2A. D2M Capoeira is 25. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $5.0 \cdot 10^{6}$ times more accurate than C2A. C2A is 71. times more accurate than D2M Capoeira. D2M is $3.6 \cdot 10^{8}$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 156. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.94 milliseconds.

K.14 Benchmark 26/27 Bunny versus Buddha (high speed)

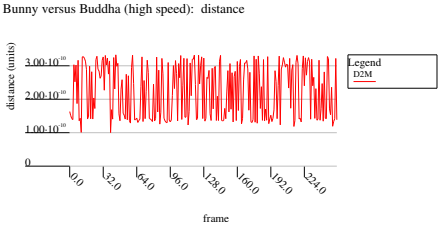

Figure 92: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	8.0	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	51.	1.0
Lower Quartile	25.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.8 \cdot 10^{2}$	1.0
Median	35.	$2.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.5 \cdot 10^{2}$	1.0
Upper Quartile	49.	$3.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.9 \cdot 10^{2}$	1.0
Maximum	$1.5 \cdot 10^{2}$	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	40.	$2.2 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.2 \cdot 10^{2}$	1.0
Standard-Deviation	23.	$7.9 \cdot 10^{-11}$	0.55	82.	0.00064

Table 40: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.50	0.00047	-	-	1.0
Lower Quartile	2.5	0.0087	-	-	1.0
Median	3.6	0.018	-	-	1.0
Upper Quartile	5.1	0.031	-	-	1.0
Maximum	11.	0.050	-	-	1.0
Mean	3.9	0.020	-	-	1.0
Standard-Deviation	1.9	0.014	-	-	0.00064

Table 41: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.16	0.0091	$5.1 \cdot 10^{2}$	-	1.0
Lower Quartile	0.82	0.018	$5.1 \cdot 10^{2}$	-	1.0
Median	1.2	0.022	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	1.6	0.032	$5.1 \cdot 10^{2}$	-	1.0
Maximum	3.7	0.052	$5.1 \cdot 10^{2}$	-	1.0
Mean	1.3	0.025	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.65	0.0089	0.0017	-	0.00064

Table 42: D2M Capoeira statistics

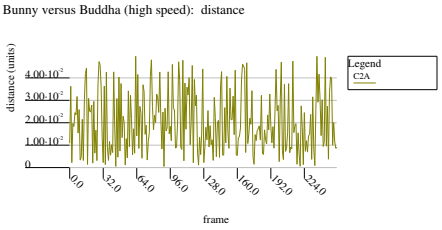

Figure 93: A plot

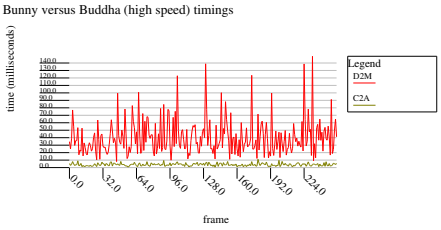

Figure 95: A dual plot

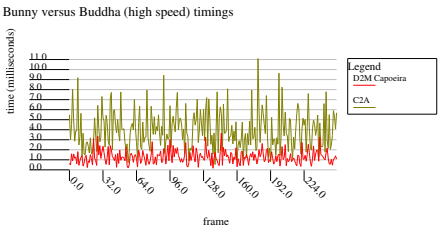

Figure 96: A dual plot

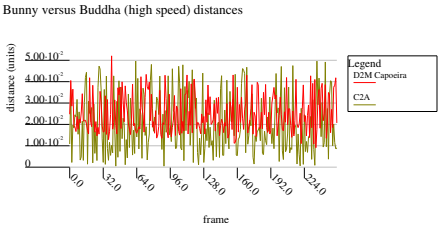

Figure 94: A dual plot

Worst Case Timing Remarks

C2A is 13. times faster than D2M. D2M Capoeira is 3.0 times faster than

C2A. D2M Capoeira is 41. times faster than D2M.

Average Timing Remarks

C2A is 10. times faster than D2M. D2M Capoeira is 3.1 times faster than C2A. D2M Capoeira is 31. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. C2A is not significantly more accurate than D2M Capoeira, nor is it less accurate than D2M Capoeira. D2M is $1.6 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.85 milliseconds.

K.15 Benchmark 28/29 Dragon versus Dragon (low speed)

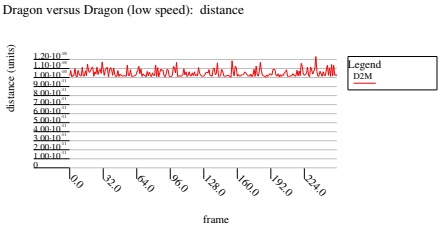

Figure 97: A plot

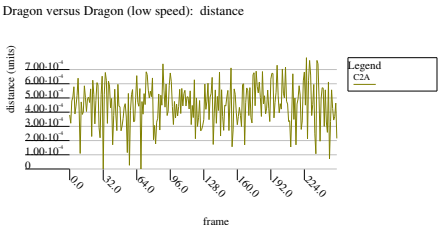

Figure 98: A plot

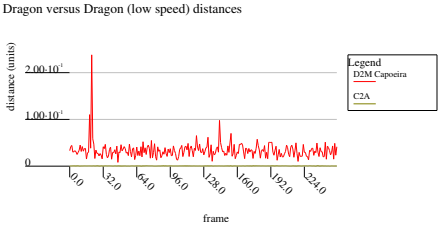

Figure 99: A dual plot

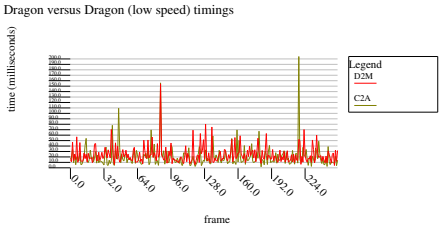

Figure 100: A dual plot

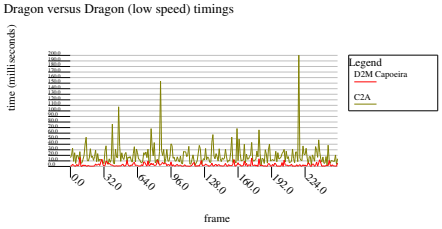

Figure 101: A dual plot

	Time	Dist.	v	v · n̂	TOC
Minimum	4.3	$1.0 \cdot 10^{-10}$	1.9	0.45	0.19
Lower Quartile	13.	$1.0 \cdot 10^{-10}$	4.3	2.9	0.34
Median	19.	$1.0 \cdot 10^{-10}$	5.0	3.8	0.42
Upper Quartile	28.	$1.1 \cdot 10^{-10}$	5.6	4.6	0.50
Maximum	$1.5 \cdot 10^{2}$	$1.2 \cdot 10^{-10}$	8.0	7.2	0.73
Mean	24.	$1.0 \cdot 10^{-10}$	4.9	3.8	0.42
Standard-Deviation	16.	$4.2 \cdot 10^{-12}$	1.0	1.2	0.11

Table 43: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	1.8	0	-	-	0.19
Lower Quartile	9.7	0.00036	-	-	0.34
Median	15.	0.00046	-	-	0.42
Upper Quartile	25.	0.00056	-	-	0.50
Maximum	$2.0 \cdot 10^{2}$	0.00078	-	-	0.73
Mean	20.	0.00045	-	-	0.42
Standard-Deviation	20.	0.00015	-	-	0.11

Table 44: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.17	0.0081	4.0	-	0.18
Lower Quartile	0.75	0.024	4.1	-	0.33
Median	1.7	0.031	4.3	-	0.40
Upper Quartile	3.4	0.040	4.8	-	0.49
Maximum	16.	0.24	5.9	-	0.70
Mean	2.8	0.034	4.5	-	0.41
Standard-Deviation	3.0	0.018	0.45	-	0.10

Table 45: D2M Capoeira statistics

Worst Case Timing Remarks

D2M is not significantly faster than C2A nor is it slower than C2A. D2M Capoeira is 13. times faster than C2A. D2M Capoeira is 9.5 times faster than D2M.

Average Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 7.3 times faster than C2A. D2M Capoeira is 8.5 times faster than D2M.

Worst Case Accuracy Remarks

D2M is $6.4 \cdot 10^{6}$ times more accurate than C2A. C2A is $3.0 \cdot 10^{2}$ times more accurate than D2M Capoeira. D2M is $1.9 \cdot 10^{9}$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 32 and 68. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 4.0 milliseconds.

K.16 Benchmark 30/31 Dragon versus Dragon (high speed)

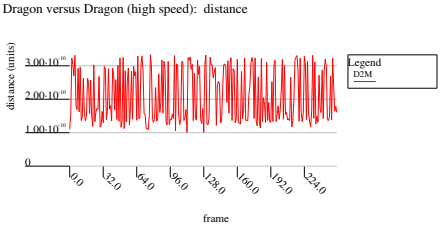

Figure 102: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	6.7	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	22.	0.99
Lower Quartile	18.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.6 \cdot 10^{2}$	0.99
Median	25.	$1.7 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.2 \cdot 10^{2}$	1.0
Upper Quartile	34.	$2.9 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.7 \cdot 10^{2}$	1.0
Maximum	$1.0 \cdot 10^{2}$	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	27.	$2.1 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.0 \cdot 10^{2}$	1.0
Standard-Deviation	14.	$7.9 \cdot 10^{-11}$	0.91	95.	0.00087

Table 46: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.78	0.0014	-	-	0.99
Lower Quartile	2.6	0.011	-	-	0.99
Median	4.0	0.020	-	-	1.0
Upper Quartile	5.8	0.031	-	-	1.0
Maximum	22.	0.050	-	-	1.0
Mean	4.7	0.022	-	-	1.0
Standard-Deviation	3.0	0.013	-	-	0.00088

Table 47: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	0.0072	$5.1 \cdot 10^{2}$	-	0.99
Lower Quartile	0.63	0.025	$5.1 \cdot 10^{2}$	-	0.99
Median	0.90	0.031	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	1.4	0.039	$5.1 \cdot 10^{2}$	-	1.0
Maximum	3.7	0.059	$5.1 \cdot 10^{2}$	-	1.0
Mean	1.0	0.032	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.62	0.0099	0.0044	-	0.00088

Table 48: D2M Capoeira statistics

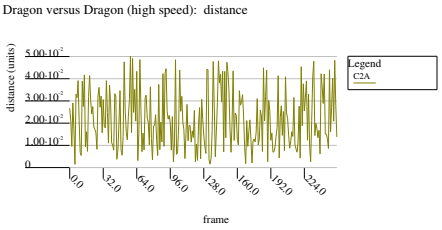

Figure 103: A plot

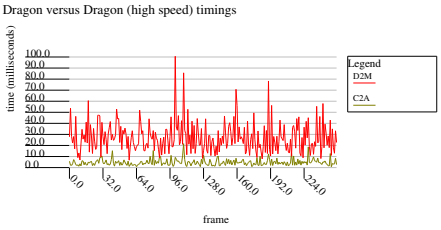

Figure 105: A dual plot

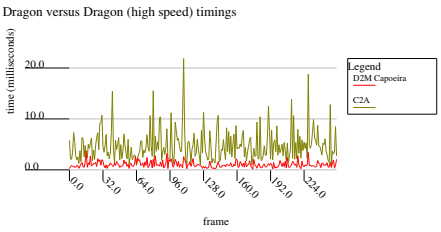

Figure 106: A dual plot

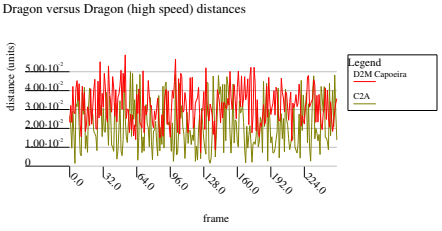

Figure 104: A dual plot

Worst Case Timing Remarks

C2A is 4.7 times faster than D2M. D2M Capoeira is 5.8 times faster than

C2A. D2M Capoeira is 27. times faster than D2M.

Average Timing Remarks

C2A is 5.8 times faster than D2M. D2M Capoeira is 4.5 times faster than C2A. D2M Capoeira is 26. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. C2A is not significantly more accurate than D2M Capoeira, nor is it less accurate than D2M Capoeira. D2M is $1.8 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.56 milliseconds.

K.17 Benchmark 32/33 Dragon versus Buddha (low speed)

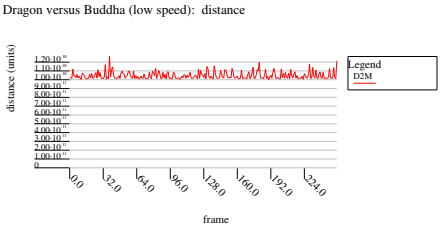

Figure 107: A plot

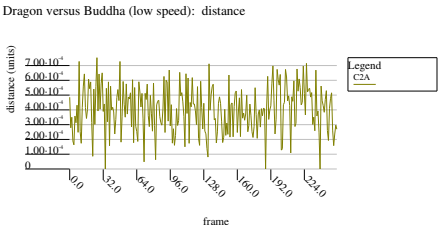

Figure 108: A plot

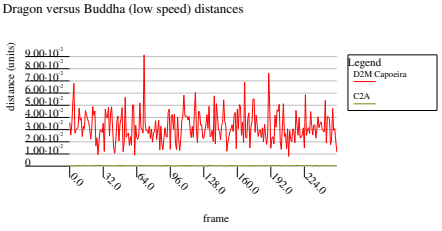

Figure 109: A dual plot

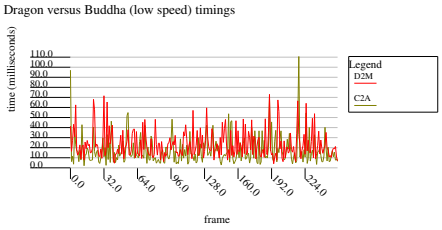

Figure 110: A dual plot

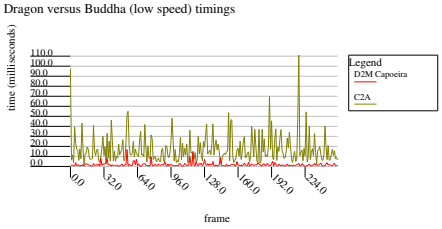

Figure 111: A dual plot

	Time	Dist.	v	v · n̂	TOC
Minimum	4.0	$1.0 \cdot 10^{-10}$	2.2	0.35	0.27
Lower Quartile	13.	$1.0 \cdot 10^{-10}$	4.1	2.9	0.46
Median	19.	$1.0 \cdot 10^{-10}$	4.7	3.7	0.53
Upper Quartile	30.	$1.1 \cdot 10^{-10}$	5.1	4.4	0.60
Maximum	73.	$1.3 \cdot 10^{-10}$	6.7	6.6	0.80
Mean	23.	$1.0 \cdot 10^{-10}$	4.6	3.6	0.53
Standard-Deviation	14.	$4.3 \cdot 10^{-12}$	0.79	1.1	0.11

Table 49: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	2.1	0	-	-	0.27
Lower Quartile	7.9	0.00029	-	-	0.46
Median	13.	0.00041	-	-	0.53
Upper Quartile	21.	0.00052	-	-	0.60
Maximum	$1.1 \cdot 10^{2}$	0.00075	-	-	0.80
Mean	17.	0.00040	-	-	0.53
Standard-Deviation	14.	0.00016	-	-	0.11

Table 50: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.24	0.0081	4.0	-	0.26
Lower Quartile	0.74	0.024	4.1	-	0.44
Median	1.2	0.031	4.2	-	0.52
Upper Quartile	2.2	0.040	4.5	-	0.59
Maximum	17.	0.091	5.4	-	0.78
Mean	1.8	0.032	4.3	-	0.52
Standard-Deviation	2.0	0.012	0.31	-	0.11

Table 51: D2M Capoeira statistics

Worst Case Timing Remarks

D2M is 1.5 times faster than C2A. D2M Capoeira is 6.6 times faster than C2A. D2M Capoeira is 4.3 times faster than D2M.

Average Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 9.5 times faster than C2A. D2M Capoeira is 13. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $6.0 \cdot 10^{6}$ times more accurate than C2A. C2A is $1.2 \cdot 10^{2}$ times more accurate than D2M Capoeira. D2M is $7.2 \cdot 10^{8}$ times more accurate than D2M Capoeira.

General Remarks

C2A is either touching or penetrating at frame 34, 187 and 239. D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.65 milliseconds.

K.18 Benchmark 34/35 Dragon versus Buddha (high speed)

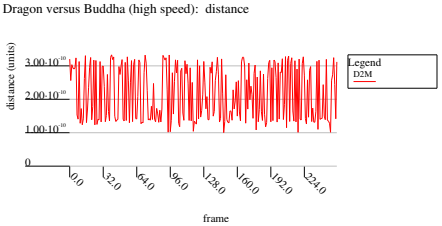

Figure 112: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	5.7	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	22.	0.99
Lower Quartile	18.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.6 \cdot 10^{2}$	1.0
Median	25.	$2.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.4 \cdot 10^{2}$	1.0
Upper Quartile	34.	$3.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.8 \cdot 10^{2}$	1.0
Maximum	$1.0 \cdot 10^{2}$	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	29.	$2.2 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.1 \cdot 10^{2}$	1.0
Standard-Deviation	15.	$8.0 \cdot 10^{-11}$	0.84	93.	0.00085

Table 52: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.69	0.00081	-	-	0.99
Lower Quartile	2.1	0.0094	-	-	1.0
Median	3.3	0.017	-	-	1.0
Upper Quartile	5.2	0.030	-	-	1.0
Maximum	13.	0.049	-	-	1.0
Mean	4.0	0.020	-	-	1.0
Standard-Deviation	2.6	0.013	-	-	0.00086

Table 53: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	0.0040	$5.1 \cdot 10^{2}$	-	0.99
Lower Quartile	0.62	0.023	$5.1 \cdot 10^{2}$	-	1.0
Median	1.0	0.030	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	1.5	0.037	$5.1 \cdot 10^{2}$	-	1.0
Maximum	3.6	0.063	$5.1 \cdot 10^{2}$	-	1.0
Mean	1.1	0.031	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.65	0.011	0.0029	-	0.00086

Table 54: D2M Capoeira statistics

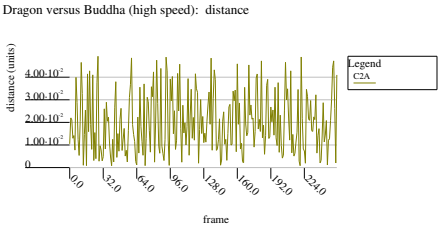

Figure 113: A plot

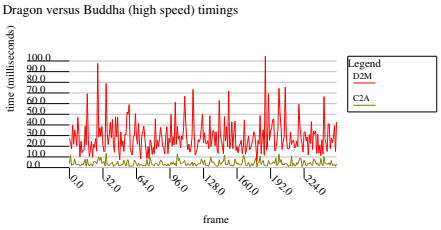

Figure 115: A dual plot

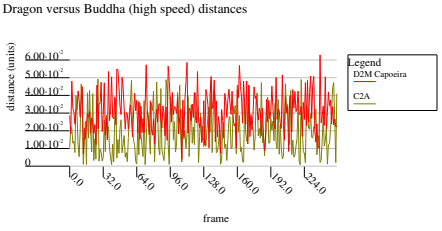

Figure 114: A dual plot

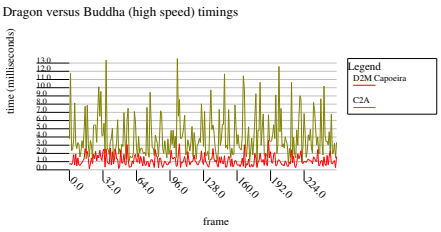

Figure 116: A dual plot

Worst Case Timing Remarks

C2A is 7.8 times faster than D2M. D2M Capoeira is 3.7 times faster than

C2A. D2M Capoeira is 29. times faster than D2M.

Average Timing Remarks

C2A is 7.1 times faster than D2M. D2M Capoeira is 3.5 times faster than C2A. D2M Capoeira is 25. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. C2A is not significantly more accurate than D2M Capoeira, nor is it less accurate than D2M Capoeira. D2M is $1.9 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.60 milliseconds.

K.19 Benchmark 36/37 Buddha versus Buddha (low speed)

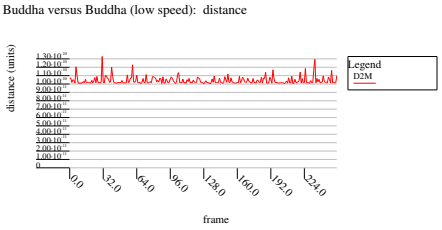

Figure 117: A plot

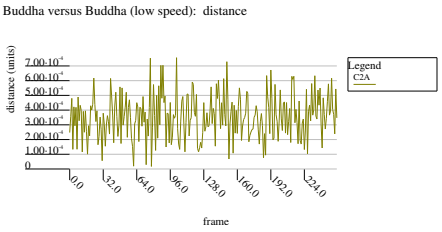

Figure 118: A plot

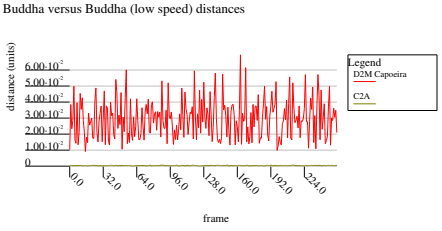

Figure 119: A dual plot

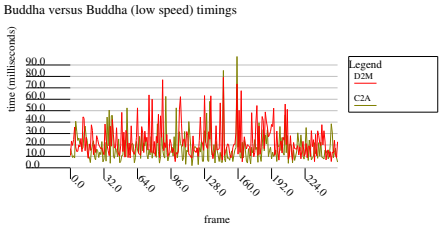

Figure 120: A dual plot

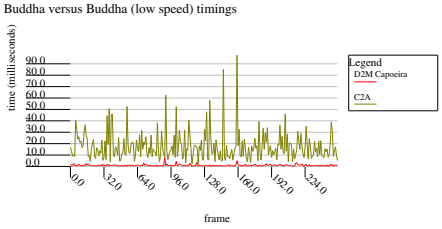

Figure 121: A dual plot

	Time	Dist.	v	v · n̂	TOC
Minimum	5.2	$1.0 \cdot 10^{-10}$	2.4	0.23	0.36
Lower Quartile	13.	$1.0 \cdot 10^{-10}$	4.0	2.9	0.58
Median	19.	$1.0 \cdot 10^{-10}$	4.4	3.5	0.66
Upper Quartile	28.	$1.1 \cdot 10^{-10}$	4.8	4.1	0.74
Maximum	79.	$1.3 \cdot 10^{-10}$	6.7	5.9	0.83
Mean	23.	$1.0 \cdot 10^{-10}$	4.4	3.4	0.65
Standard-Deviation	14.	$4.6 \cdot 10^{-12}$	0.68	0.89	0.11

Table 55: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	1.9	$1.7 \cdot 10^{-5}$	-	-	0.36
Lower Quartile	9.0	0.00025	-	-	0.58
Median	15.	0.00036	-	-	0.66
Upper Quartile	21.	0.00045	-	-	0.74
Maximum	97.	0.00076	-	-	0.83
Mean	17.	0.00036	-	-	0.65
Standard-Deviation	13.	0.00015	-	-	0.11

Table 56: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.12	0.0091	4.0	-	0.34
Lower Quartile	0.56	0.018	4.0	-	0.55
Median	0.85	0.029	4.1	-	0.65
Upper Quartile	1.2	0.037	4.3	-	0.73
Maximum	7.8	0.069	5.3	-	0.82
Mean	1.0	0.030	4.2	-	0.64
Standard-Deviation	0.79	0.012	0.22	-	0.11

Table 57: D2M Capoeira statistics

Worst Case Timing Remarks

D2M is not significantly faster than C2A nor is it slower than C2A. D2M Capoeira is 12. times faster than C2A. D2M Capoeira is 10. times faster than D2M.

Average Timing Remarks

C2A is not significantly faster than D2M nor is it slower than D2M. D2M Capoeira is 17. times faster than C2A. D2M Capoeira is 23. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $5.7 \cdot 10^6$ times more accurate than C2A. C2A is 92. times more accurate than D2M Capoeira. D2M is $5.2 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly more accurate than D2M Capoeira. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.70 milliseconds.

K.20 Benchmark 38/39 Buddha versus Buddha (high speed)

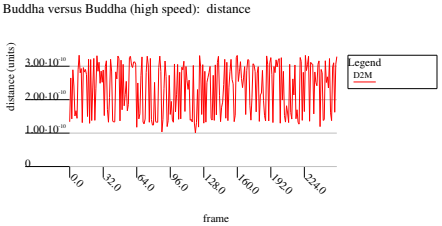

Figure 122: A plot

	Time	Dist.	v	v · n̂	TOC
Minimum	7.9	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	94.	1.0
Lower Quartile	18.	$1.4 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$3.4 \cdot 10^{2}$	1.0
Median	26.	$2.5 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.2 \cdot 10^{2}$	1.0
Upper Quartile	36.	$3.0 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.7 \cdot 10^{2}$	1.0
Maximum	80.	$3.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$5.1 \cdot 10^{2}$	1.0
Mean	28.	$2.3 \cdot 10^{-10}$	$5.1 \cdot 10^{2}$	$4.0 \cdot 10^{2}$	1.0
Standard-Deviation	13.	$7.6 \cdot 10^{-11}$	0.72	89.	0.00076

Table 58: D2M statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.67	0.00045	-	-	1.0
Lower Quartile	2.6	0.0092	-	-	1.0
Median	3.9	0.019	-	-	1.0
Upper Quartile	5.3	0.032	-	-	1.0
Maximum	15.	0.049	-	-	1.0
Mean	4.2	0.021	-	-	1.0
Standard-Deviation	2.1	0.014	-	-	0.00075

Table 59: C2A statistics

	Time	Dist.	v	v · n̂	TOC
Minimum	0.18	0.0073	$5.1 \cdot 10^{2}$	-	1.0
Lower Quartile	0.54	0.021	$5.1 \cdot 10^{2}$	-	1.0
Median	0.82	0.029	$5.1 \cdot 10^{2}$	-	1.0
Upper Quartile	1.1	0.036	$5.1 \cdot 10^{2}$	-	1.0
Maximum	2.2	0.067	$5.1 \cdot 10^{2}$	-	1.0
Mean	0.87	0.030	$5.1 \cdot 10^{2}$	-	1.0
Standard-Deviation	0.44	0.011	0.0017	-	0.00076

Table 60: D2M Capoeira statistics

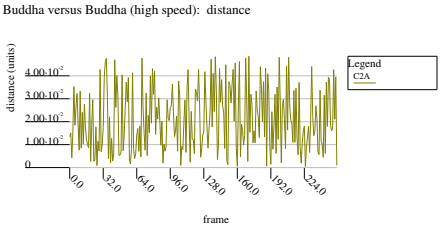

Figure 123: A plot

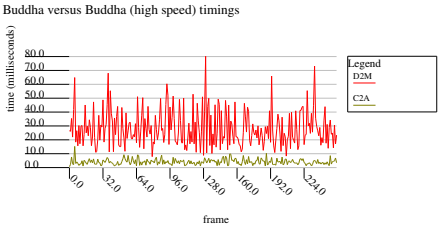

Figure 125: A dual plot

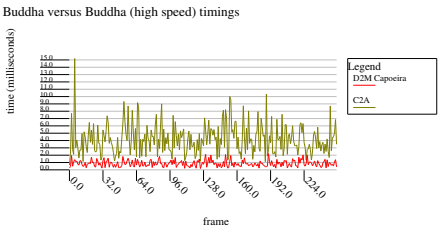

Figure 126: A dual plot

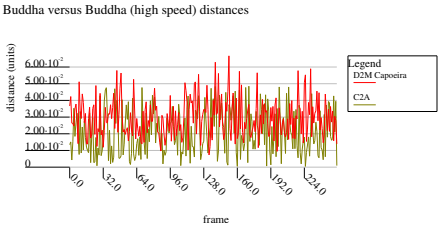

Figure 124: A dual plot

Worst Case Timing Remarks

C2A is 5.3 times faster than D2M. D2M Capoeira is 6.9 times faster than

C2A. D2M Capoeira is 36. times faster than D2M.

Average Timing Remarks

C2A is 6.8 times faster than D2M. D2M Capoeira is 4.8 times faster than C2A. D2M Capoeira is 33. times faster than D2M.

Worst Case Accuracy Remarks

D2M is $1.5 \cdot 10^8$ times more accurate than C2A. C2A is not significantly more accurate than D2M Capoeira, nor is it less accurate than D2M Capoeira. D2M is $2.0 \cdot 10^8$ times more accurate than D2M Capoeira.

General Remarks

D2M is significantly more accurate than C2A. C2A is significantly faster than D2M. D2M Capoeira is significantly faster than C2A. The worst case recorded absolute timing error of C2A is 0.61 milliseconds.

K.21 General Remarks about all Benchmarks

Note that all C2A benchmarks have been done twice; one time against D2M and one time against D2M Capoeira. Because the computations are exactly the same, the timings of these two benchmarks are interchangeable. However, the worst case deviation of a timing that is known is 4.0 milliseconds. This may be due to interference of the operating system. If a timing is measured twice, on average there is a factor 1.027 difference between them. Again, the following statistics are based on two measurements of C2A for each benchmark: The relative error in the median of a benchmark may be a factor of 1.028. The lower and upper quartiles have may have a relative error of 1.156 and 1.028 respectively. The maximum may have a relative error of 1.085. Note that the statistics above are not strict upper bounds, but they are likely to give an upper bound on the amount of deviation. The deviation should not have a considerable influence on the results.

C2A penetrates or touches in 15 of 5120 collision instances. D2M penetrates or touches in 0 of 5120 collision instances. D2M Capoeira penetrates or touches in 0 of 5120 collision instances.

	Time	**Dist.**
Minimum	0.16	$1.0 \cdot 10^{-10}$
Lower Quartile	11.	$1.0 \cdot 10^{-10}$
Median	21.	$1.1 \cdot 10^{-10}$
Upper Quartile	35.	$1.8 \cdot 10^{-10}$
Maximum	$2.6 \cdot 10^{2}$	$3.6 \cdot 10^{-10}$
Mean	26.	$1.6 \cdot 10^{-10}$
Standard-Deviation	24.	$7.7 \cdot 10^{-11}$

Table 61: D2M statistics

	Time	**Dist.**
Minimum	0.072	0
Lower Quartile	2.4	0.00035
Median	5.9	0.00060
Upper Quartile	15.	0.016
Maximum	$3.3 \cdot 10^{2}$	0.050
Mean	12.	0.0095
Standard-Deviation	19.	0.013

Table 62: C2A statistics

	Time	**Dist.**
Minimum	0.058	0.0040
Lower Quartile	0.45	0.021
Median	0.85	0.031
Upper Quartile	1.5	0.042
Maximum	22.	0.24
Mean	1.2	0.033
Standard-Deviation	1.3	0.015

Table 63: D2M Capoeira statistics