# OpenCollBench - Benchmarking of Collision Detection & Proximity Queries as a Web-Service

Toni Tan
University of Bremen
Germany

René Weller
University of Bremen
Germany

Gabriel Zachmann
University of Bremen
Germany

## ABSTRACT

We present a server-based benchmark that enables a fair analysis of different collision detection & proximity query algorithms. A simple yet interactive web interface allows both expert and non-expert users to easily evaluate different collision detection algorithms' performance in standardized or optionally user-definable scenarios and identify possible bottlenecks. In contrast to typically used simple charts or histograms to show the results, we additionally propose a heatmap visualization directly on the benchmarked objects that allows the identification of critical regions on a sub-object level. An anonymous login system, in combination with a server-side scheduling algorithm, guarantees security as well as the reproducibility and comparability of the results. This makes our benchmark useful for end-users who want to choose the optimal collision detection method or optimize their objects with respect to collision detection but also for researchers who want to compare their new algorithms with existing solutions.

## CCS CONCEPTS

• **Computing methodologies → Collision detection**; **Shape analysis**.

## KEYWORDS

collision detection, proximity query, open benchmark, heatmap visualization, semantic information, benchmark as web-service

**ACM Reference Format:**
Toni Tan, René Weller, and Gabriel Zachmann. 2020. OpenCollBench - Benchmarking of Collision Detection & Proximity Queries as a Web-Service. In *The 25th International Conference on 3D Web Technology (Web3D '20), November 9–13, 2020, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3424616.3424712

## 1 INTRODUCTION

*Collision detection (CD)* is essential in many applications, such as physically-based simulation, motion planning, and computer games. In many of these applications, CD is the computational bottleneck. For instance, in randomized path planners, more than 90% of the computation time is spent on collision detection [Hsu et al. 1998]. The most time-consuming part is usually the so-called *narrow phase* CD, i.e., detecting whether a pair of 3D objects intersect or not. Closely related to this challenge are *proximity queries (PQ)* that additionally report the minimum distance between the pair of objects in case of non-collision.

Due to the wide variety of use cases and inherent complexity of the problem, CD & PQ have been researched since several decades in different communities, and they remain an active field of research because until now, no all-in-one algorithm suitable for every purpose has been found if such a solution could exist at all.

Most available CD & PD algorithms for the narrow phase rely on bounding volume hierarchies (BVHs) to accelerate the queries. The idea is, instead of calculating slow and complex tests on the geometric primitives, objects are enclosed recursively with simple bounding volumes (BV) that allow culling parts of the geometry to avoid further testing. Many different bounding volumes have been proposed to build such BVHs for CD & PD, including spheres [Hubbard 1996], AABBs [van den Bergen 1998] [Zachmann 1995], k-DOPs [Klosowski 1998] [Zachmann 1998], OBBs [Gottschalk et al. 1996], spherical shells [Krishnan et al. 1998], swept spheres [Larsen et al. 1999a], to name but a few. Moreover, BVHs can have different branching factors, the BVHs can be constructed in different ways (e.g., iteratively, bottom-up, or top-down), the primitives can be assigned in different ways to the BVs in the hierarchy (for instance, via middle split, median split or even using sophisticated clustering algorithms) and finally, there exist different algorithms for the hierarchy traversal during run-time [Tan et al. 2019].

The reason for such a large amount of different CD & PQ approaches is that they are often optimized for a particular scenario. CD & PQ algorithms are usually susceptible to certain factors like relative the object's shape (e.g., convex or concave), the sizes between objects, relative distances, the sizes, shapes, and distributions of the geometric primitives or the transformations between objects, to name but a few. Moreover, the limitations of the algorithms are hardly discussed in the publications, if actually known. In many publications, authors usually use a set of self-defined objects & scenarios to benchmark & compare their proposed algorithms with existing ones. However, this is not always in favor of existing algorithms since authors might choose objects or scenarios that favor their proposed algorithms. Even more, the source code of competing algorithms is often unavailable or outdated, and there is no access to objects and scenarios used by the competing algorithms for their benchmarks. Besides that, technical difficulties, i.e., the sheer amount of involved parameters or integration of existing CD algorithms making benchmarking of CD algorithms a complicated and time-consuming process. Finally, the reported scenarios often only show an average, sometimes a standard deviation, and maybe the maximum running time for a whole sequence of transformations. This is not sufficient to understand *why* a certain algorithm

*Web3D '20, November 9–13, 2020, Virtual Event, Republic of Korea*
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8169-7/20/11…$15.00
https://doi.org/10.1145/3424616.3424712

performs better or worse in a particular scenario. Actually, a slight change of transformations or the objects, e.g., a slightly different polygonization of the object, could result in completely different results.

In this paper, we present an idea to simplify the complex and time-consuming process of benchmarking collision detection algorithms, more precisely, of CD methods for the broad phase CD of rigid polygonal polygon soups. Moreover, we provide a set of predefined scenarios, i.e., a set of objects together with configurations that cover a broad range of interesting collision detection cases. However, this set can also be extended by the users to include scenarios that we did not consider. If allowed by the user, these new scenarios can be included in the benchmark and will be made available to the public.

The main idea is to provide the benchmarking of CD & PQ as an online service. This has the advantage that a large amount of collision detection algorithms is available as pre-compiled libraries on a common, unified hardware platform via an easy-to-use but nevertheless highly adjustable web interface. Additionally, the extending object and configuration database allow us to cover an increasing number of interesting collision scenarios. This web-based service facilitates the comparison of CD algorithms dramatically and is of interest to both users of CD algorithms who simply want to find the best choice for their particular scenario and CD researchers, who want to compare their new algorithms to competitors.

Our web-based service provides a front end interface that allows the users to adjust some benchmark parameters, e.g., selecting scenarios, algorithms, or upload their objects and generate a set of configurations. The actual benchmark is performed on a dedicated back end server PC that is reserved for only this task in order to not disturb the benchmarking procedure by simultaneous web access and, obviously, for security reasons. All benchmarks are scheduled to guarantee the same computational power for all users.

The basis of our web service is a well established benchmarking suite for collision detection algorithms [Trenkel et al. 2007]. It has a well defined and easy-to-use interface to include new algorithms, and it already delivers a set of interesting collision scenarios. However, we further extended it to also support proximity queries instead of simple boolean collision queries. Moreover, we heavily extended its' analyzation functionalities: the original benchmarking suite simply computes the average and maximum collision detection times and plots them to charts or histograms. Our web service offers the possibility to overlay the 3D object with a detailed heatmap. This facilitates it to identify interesting object regions, e.g., regions that are hardly checked for collisions, regions where particular algorithms perform better or worse, etc.

We are confident that this new method to visualize information from the collision detection benchmark will influence the further research of collision detection, for instance, by optimizing BVH construction algorithms or by optimizing the geometry for particular CD algorithms. Moreover, we think that the general idea of providing benchmarking as a web service can be also interesting for other research fields and is an interesting research field for its own, e.g., with respect to the user interface or the display of the results in 3D, perhaps directly projected as a heatmap to 3D objects.

This could benefit both expert and non-expert users in many real applications, i.e., choosing optimal CD algorithms according to use case or optimizing objects for CD by removing/modifying slow regions.

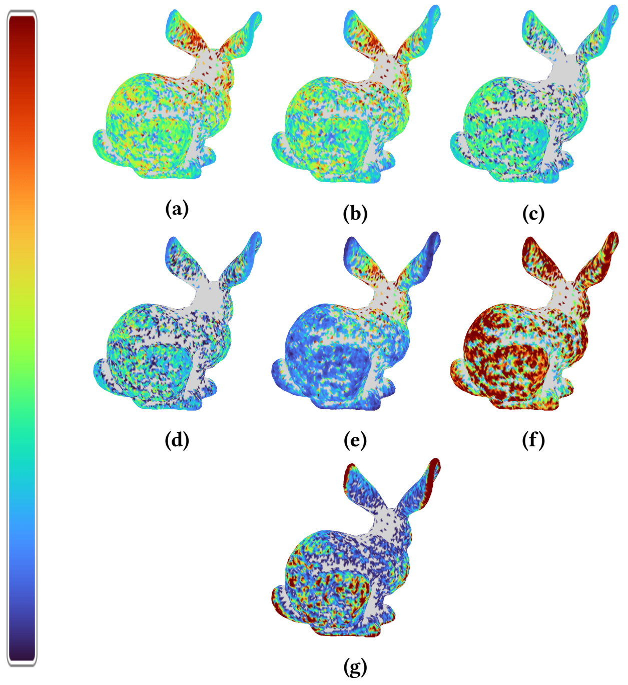


**Figure 1: An example of heatmap based on configuration's (a) average timings, (b) median timings, (c) standard deviation timings, (d) median absolute deviation timings, and (e) min timings, (f) max timings, and (g) density.**

## 2 RELATED WORK

The benchmarking process varies across different fields, i.e., multi-object tracking [Dendorfer et al. 2020] compared the result of the proposed algorithm against ground truth annotated by a human. In ray tracing, benchmarking is usually done using a set of predefined scenarios, e.g., the Benchmark for Animated Ray Tracing (BART) [Lext et al. 2001].

Benchmarking programs are typically provided as standalone programs, which can restrict access due to hardware or software constraints. An attempt to solve this is to offer benchmarking suites as web-service [Gillard and Vandenbosch 2009] [Widlowski et al. 2008]. To our knowledge, this idea was never applied for computer graphics related topics, especially on an algorithmic level that would help users to choose the best algorithm for their specific scenario and supports developers and researchers with an infrastructure for optimizing and distributing their algorithms. Actually, there exist different graphics algorithms that could benefit from such an online service. We decided to choose the complex problem of collision detection because there exists a variety of different algorithms, and CD is often the computational bottleneck.

Usually, authors of collision detection simply define a certain scenario on their own to test their algorithms. For instance, Otaduy et al. [Otaduy and Lin 2003] used a set of self-defined scenarios (wrinkled torus falling along with a spiral peg, spoon sliding inside a cup, soup of numbers settling in a bowl) to benchmark their proposed CD algorithm. Van Den Bergen [van den Bergen 1998] positioned a pair of objects inside a bounded space randomly and tested them for the intersection. The probability of intersection is controlled by changing the size of objects.

There exist only very few efforts to provide general, fair, and reproducible benchmarks for CD algorithms. Zachmann [Zachmann 1998] proposed a simple benchmark for DopTree that also applies to general algorithms by positioning two identical objects at a certain distance relative to each other. The relative distance is calculated based on the center of the object's bounding box. One object will stay still, while the other performs a full rotation around the z-axis at fixed small steps. The average timing of the CD algorithm is calculated by averaging CD time at all steps.

Caselli et al. [Caselli et al. 2002] use several predefined scenes in a probabilistic motion planner to benchmark several advanced collision detection algorithms, i.e., V-Clip, RAPID, SOLID, PQP, and V-Collide. However, the results can not be directly transferred to scenarios not included in the benchmark.

Trenkel et al. [Trenkel et al. 2007] proposed a systematic way to measure CD algorithms by combining broad and narrow phases from Hubbard [Hubbard 1993] taxonomies into a CD pipeline. The test scenarios are generated by positioning two identical objects at a predefined distance. The positions and orientations for the predefined distance are generated by rotating and translating one of the objects.

Diktas et al. [Diktas and Sahiner 2008] argue that it is not enough to test algorithms based on the relative distance between objects since objects might penetrate against each other. They proposed a benchmarking suite that takes relative penetration along with relative distance and size into account. They presented a way to generate a position by performing continuous CD using sphere-tree fitted to object against the object's surface offset.

Weller et al. [Weller et al. 2010] extended [Trenkel et al. 2007] to include relative penetration between objects. They proposed a method to measure the quality of force and torque for 6 DOF (Degrees Of Freedom) haptic rendering and applied it to evaluate two algorithms, i.e., Voxmap-Pointshell (VPS) and Inner Sphere Tree (IST).

Woulfe et al. [Woulfe and Manzke 2009] proposed a generic benchmarking suite for interactive applications. They enable users to supply parameters that mimic the standard geometric and physical properties of rigid bodies, i.e., position, size, mass, acceleration, velocity, etc. However, it is limited to CD algorithms available in Bullet Physics. The object is also predefined, which makes it impossible to test a custom object. Besides that, adding a custom CD algorithms into Bullet is extremely difficult.

Although there exist some efforts to provide a fair and systematic benchmark for CD algorithms, little to none work has been put to provide a better understanding of benchmarking results on a *sub-object* level to identify, for instance, parts of an object that are maybe especially well or badly suited for a certain algorithm. Results are mostly represented using a chart or histogram based on algorithms' average or maximum timings for the whole sequence of configurations and a complete pair of objects, which is not sufficient to understand CD algorithms' behavior & characteristic in-depth.

## 3 OUR APPROACH

Our OpenCollBench consists of three parts: an easy-to-use but highly adjustable benchmark for CD and PQ algorithms, a novel visualization method for the results of the benchmarks that supports a sophisticated but understandable inspection of the results even for inexperienced users on a sub-object level, and a web-based system that offers our benchmark as a service. In the following, we will detail the individual parts of OpenCollBench, starting with the actual benchmark.

### 3.1 Collision Detection Benchmark

The core benchmarking functionality of OpenCollBench relies on an already available standardized open-source benchmarking suite by Trenkel et al. [Trenkel et al. 2007]; hence we will start with a very quick recap. In general, the benchmarking suite is a suitable narrow phase CD of arbitrary polygonal rigid objects, and it supports even polygon soups. It is based on the observation that the running-time of boolean CD algorithms is worst in the case that the objects are in close proximity but do not collide: in this case, the typical BVH-based algorithms have to traverse very often down to the leaves, but they cannot stop the traversal because they do not find an actual intersection; hence a lot of backtracking is necessary by the recursive traversal algorithms. Moreover, it relies on the assumption that in interactive applications, it is not known in advance in which particular configuration, i.e., translation and orientation, the pair of objects will collide; hence, we have to consider all of them.

As a consequence, the benchmarking suite samples the configuration space with a user-definable accuracy. The sampling includes the possible orientations and distances of the objects. Two different sampling methods are available; one simply places one object on a sphere and moves it towards the second object until a certain distance is met, whereas the second method uses a grid for the initial positioning of the moving object. The second method is more accurate but requires more computation time to generate a set of configurations. The user can define the set of objects. A set of objects in different polygonal resolutions and pre-computed configurations for these objects is available. For more details, we refer the interested reader to [Trenkel et al. 2007].

The benchmark offers a lightweight and well-documented C++ interface: developers simply have to write a small wrapper that offers two functions, one to import the polygonal model and a second one to move the objects according to a 4x4 transformation matrix. There already exist many wrappers for current state-of-the-art collision detection libraries like CollDet with its different included algorithms, including the new SIMDop [Tan et al. 2019] that uses SIMD units of modern CPUs for the acceleration of the traversal, PQP [Larsen et al. 1999b], DOPTree [Zachmann 1998], BoxTree [Zachmann 1995], and V-COLLIDE [Hudson et al. 1997]. The benchmark is based on OpenSG, and this has the advantage that it supports a lot of different 3D object formats to be imported. Moreover, it has a headless mode, which is essential for server operation and guarantees benchmarking results that are not disturbed by interferences with the graphical output. In headless mode, all the parameters can be passed to the benchmark via the command line.

In its original form, the benchmarking suite supports only boolean collision detection algorithms, i.e., algorithms that tell whether a pair of objects collide or not. We have extended it also for proximity queries. In this case, the algorithms report minimum distances in case of non-penetrations. This kind of information is often required in robotic applications such as path planning. We only slightly changed the wrapper interfaces for algorithms that also support distance computations; the configuration generation remained untouched. Moreover, we extended the data that is collected during a benchmarking procedure that we will use in the next section for our heatmap visualization: the main difference is that we count for each triangle how often it appears in a polygon test, and we count the number of bounding volume and polygon tests for each configuration.

## 3.2 Heatmap Visualization

The benchmarking suite by Trenkel et al. [Trenkel et al. 2007] already includes several scripts based on Gnuplot to generate plots of the results: for instance, for a pair of objects at a certain polygon count, it can plot the average or maximum running time of the benchmarked algorithms with respect to the distance, or it can plot the running-time with respect to polygon count for a fixed distance. Such plots are useful to get a broad overview of the algorithms' performance with a particular pair of objects. However, depending on the object, it is possible, that the maximum running time is realized only at a very special part of the object that is hardly colliding in the target application. Even more, maybe a slight change of the object, e.g., placing an antenna a few polygons to the right or the left, might change the performance of the collision detection dramatically, so can also do a simple re-polygonization of parts of the object. Consequently, we decided to implement a novel, more sophisticated visualization of the benchmarking results on a sub-object level. The main idea is to visualize different results directly on the object's surface by using a heatmap.

To do that, we collect additional data, as written in the previous section, during the benchmark. For a pair of 3D objects $A$ and $B$ and a set $C$ of $n$ configurations $C = (c_1, c_2, ..., c_m)$ that was generated by the benchmarking suite, we store for each configuration $c_i \in C$ the collision check time $t_i$, the number of tested bounding volumes $bv_i$, the number of tested polygons $n_i$. Then we project the data to the object to generate the heatmap. Therefore, we compute for each configuration $c_i$ the closest point $p_i$ between the pair of objects (see Figure 2b). This is usually located on a polygon $p$ of $A$ and one $B$. In order to generate a heatmap for $A$ we assign the measured values $t_i$, $bv_i$, and $n_i$ to all vertices of $p$. Obviously, we normalize the assigned vertex values by dividing them by the number of assignments.

This facilitates it to identify interesting object regions, e.g., regions that are hardly checked for collisions, regions where particular algorithms perform better or worse, etc.

These vertex values can be easily mapped to color values when showing the heatmaps in our web GUI. We support different mappings of the values to colors, namely:

(1) Average (Figure 1a), median (Figure 1b), min (Figure 1e), and max (Figure 1f) timing.
    - to visualize critical regions based on algorithm's timing.

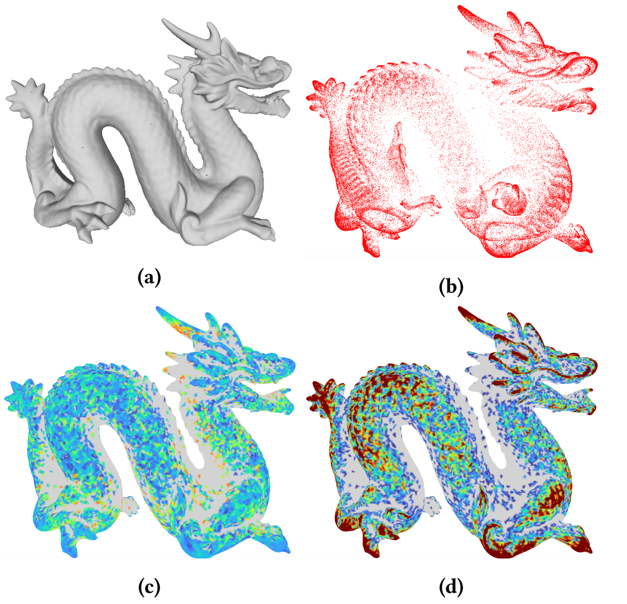


**Figure 2: Heatmap generation pipeline based on benchmark's result: (a) 3D object, (b) closest points of all configurations, (c) generated heatmap based on algorithm timings, and (d) generated heatmap based on configurations density.**

(2) Standard Deviation (Figure 1c) and Median Absolute Deviation (Figure 1d)
    - to visualize outlier regions where algorithm's timing could differs greatly between slightly different configurations.

(3) Configuration density (Figure 1g)
    - to visualize regions that are extensively or hardly checked by algorithms.

We also support an optional outlier removal based on the interquartile range (see Figures 3). Using $t_i$, $bv_i$, and $n_i$ the heatmaps can be generated to visualize the average or median time and also another statistical information to classify the data like the standard deviation (see Figures 2c and 2d for some examples), as well as the number of tested polygons (see Figure 4a), and the number of BV checks (see Figure 4b).

## 3.3 Web-based Benchmarking Service

A primary goal of OpenCollBench as a benchmark as a service is to simplify the time-consuming process of integrating CD and configuring algorithms and to provide a common hard- and software platform to produce long-term reproducible and comparable results. We have realized this by a web-based client-server architecture. Figure 6 shows an overview of our system; it is based on a front end that provides an easy-to-use GUI to the user and a dedicated back end server that performs the actual benchmarking.

The front end is designed to focus on simplification and usability of the benchmarking process to enable both expert and non-expert users to intuitively benchmark CD algorithms. We have implemented our front end using the vue.js framework. Figure 7 shows the website to select appropriate benchmark parameters via sliders

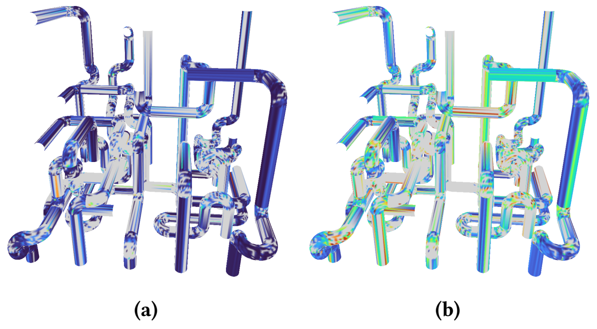


**Figure 3: Heatmaps of object pipes with 124k polygons based on median value (timing in milisec) of 200k configurations (a) before, and (b) after removing outliers using interquartile range.**

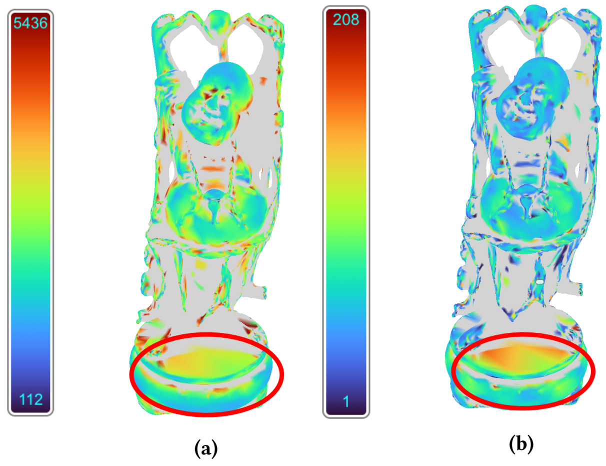


**Figure 4: Heatmaps of object happy buddha with 100k polygons based on statistical information of 200k configuration's density (the number of check) (a) BV check, and (b) polygon check of DOPTree algorithm. The red circle shows regions that heavily checked at both BVH and primitive level.**

and buttons. Additionally, it is possible to upload objects and optionally store them together with the generated configurations. Another option is to register for an account to recall previous benchmarking results or re-trigger past benchmark runs. In order to prevent failed benchmark due to connection problem or time constraints, we mark incoming benchmark requests with a unique id and store the id to the user's browser locally via cookies. This request-id enables the user to resume ongoing benchmarks. We also implemented a progress interface (see Figure 8) to keep the user informed about ongoing benchmark, e.g., uploading objects, generating configuration, performing benchmark, or generating heatmaps. By default, all generated results will be saved on our server for a period of time in case the same object is being tested again. However, we plan to add a more sophisticated access system that optionally allows users to secure their uploaded objects and results in the future. This is required, especially for industrial users that wish nondisclosure. Traditional plots of the results of the benchmark can be downloaded. Moreover, our client offers the possibility to inspect the objects with the heatmap overlay discussed in the previous section. The visualization is realized in WebGL via three.js. The heatmap viewer can be adjusted by the user to show the different results, switch outlier removal on and off, or chose an appropriate coloring method (see Figure 9).

The front end communicates via Axios with our dedicated back end server. In general, our back end server is implemented using the Express framework on top of node.js. It consists of several modules:

- *Request handler* handles incoming benchmark requests. It also assigns the unique id and schedules the requests via a queue system to prevent benchmarking suites from running multiple instances at one time, which will mess up CD algorithms' timing. The request handler is implemented with express.js.
- *Collision Benchmarking Suite* performs the actual CD & PQ benchmark for a given object and parameters. It also is responsible for generating the configurations according to the user's selected parameters. The benchmarking suite is implemented in C++, and it uses OpenSG, according to Trenkel et al. [Trenkel et al. 2007].
- *Heatmap Generation Pipeline* generates the heatmaps, i.e., the vertex colors, from the benchmark results. It is implemented in implemented using three.js.
- *Exporter* finally exports the generated heatmap into a file for further access by the front end.

Our server runs under Windows 10 on an Intel i9-9820X CPU with 10 discrete CPU cores; Hyperthreading is enabled to support 20 Threads, 64GB RAM, and GTX 980 GPU. Currently, none of the included algorithms uses multithreading for the narrow phase collision detection. The Intel Turbo Boost is enabled, this allows single-core applications to increase the maximum CPU frequency. We decided to use a current state-of-the-art Intel CPU because, in contrast to the recent AMD CPUs, it supports the most advanced SIMD acceleration technique, which is at the moment AVX512. Our results show that collision detection algorithms can benefit from this technology dramatically. On the other hand, the GPU seems a little bit outdated. However, most available narrow phase CD & PQ algorithms for rigid objects, particularly all of the algorithms currently supported by the benchmarking suite, run completely on the CPU. Moreover, the benchmark runs in headless mode; hence, there is no need for a powerful GPU at the moment. Obviously, in the future, the state-of-the-art in both hardware and software might change. In the case of large development steps, we will have to replace our current server. In order to still guarantee comparable results, we will simple re-trigger all benchmarks that are stored so far and update the results. The users will be informed about the new results automatically if they agree to this procedure. Moreover, in the case that submitted CD libraries do not work on a new platform, we will contact the developers to adjust their libraries or exclude them from further benchmarking. This will motivate developers to maintain their software to be further included in the benchmarking suite and hence, to be considered by those users searching for an appropriate CD solution and to be cited in future applications.

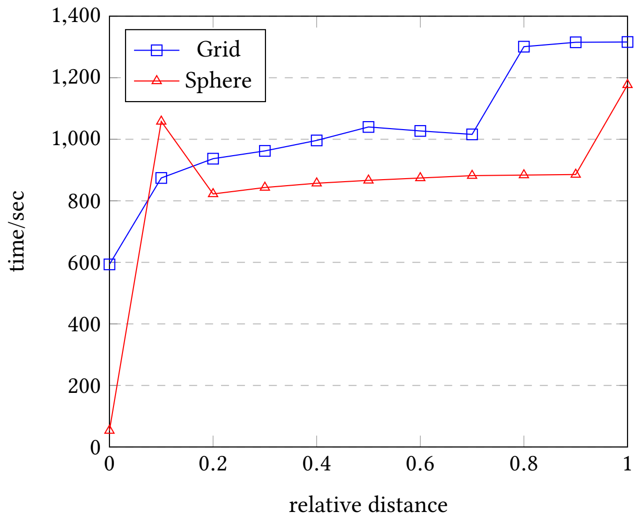


**Figure 5: Time needed to generate around 200k configurations based on *Grid* and *Sphere* position finding methods for object bunny with 65k polygons at various predefined distances. The position finding methods tend to be slower at the higher distance between objects.**

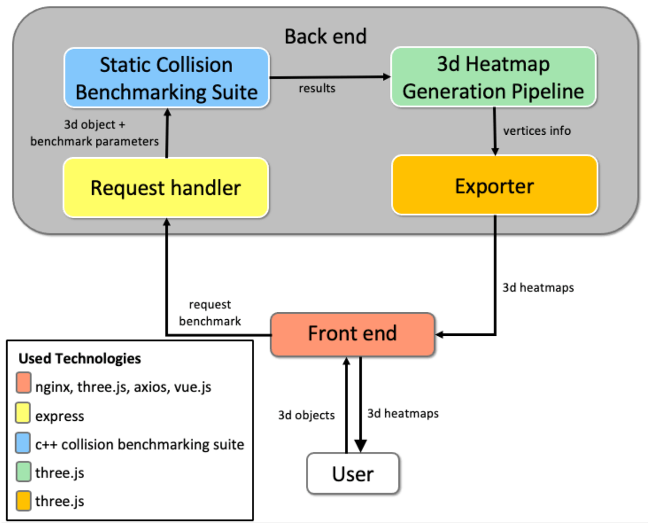
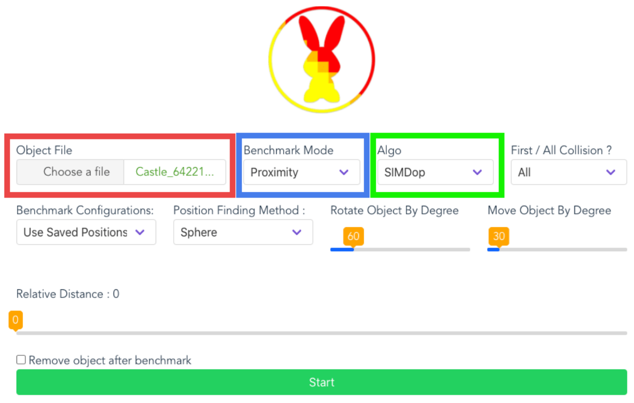

Back end
Static Collision Benchmarking Suite
results
3d Heatmap Generation Pipeline
3d object + benchmark parameters
vertices info
Request handler
Exporter
3d heatmaps
request benchmark
Front end
3d objects
3d heatmaps
User
Used Technologies
nginx, three.js, axios, vue.js
express
c++ collision benchmarking suite
three.js
three.js

**Figure 6: System Overview of OpenCollBench, which consists of two parts, namely *front end* that enable user to upload 3d object and select benchmarking parameters, and *back end* that process incoming benchmarking request and return heatmap as result.**

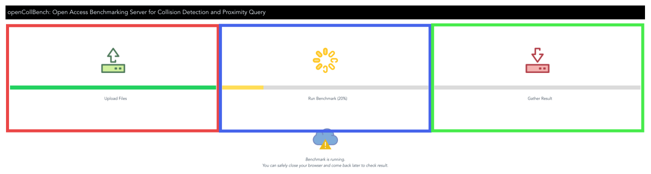
## 4 RESULTS

We have implemented our open benchmarking server as a web service to allow both expert and non-expert users to easily evaluate CD & PQ algorithms' performance in standardized or optionally user-definable scenarios and to identify possible bottlenecks. The web service is open for the public and can be accessed at URL:



Object File: Castle_64221...
Benchmark Mode: Proximity
Algo: SIMDop
First / All Collision ?: All
Benchmark Configurations: Use Saved Positions
Position Finding Method : Sphere
Rotate Object By Degree: 60
Move Object By Degree: 30
Relative Distance : 0
Remove object after benchmark

**Figure 7: Interactive Graphical User interface (GUI) for OpenCollBench, which enable user to upload object (red box) and selecting benchmark parameters interactively. The option panels connected with each other, i.e. changing *Bench Mode* (blue box) to proximity will display algorithms that support promxity query in *Algo* (green box).**



openCollBench: Open Access Benchmarking Server for Collision Detection and Proximity Query

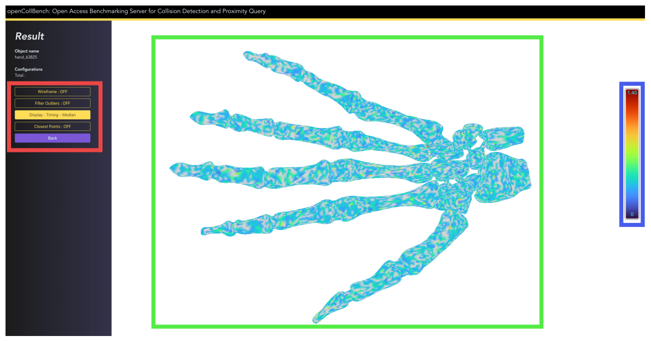
Upload Files

Run Benchmark (20%)

Gather Result

*Benchmark is running.*
*You can safely close your browser and come back later to check result.*

**Figure 8: Benchmark's progress GUI for OpenCollBench, which consists of three parts, namely, *left* (red box) showing progress of object uploaded by user, *middle* (blue box) showing benchmarking progress including configurations generation, and *right* (green box) showing progress of heatmap generation pipeline.**



openCollBench: Open Access Benchmarking Server for Collision Detection and Proximity Query

*Result*

Object name
hand_63825

Configurations
Total :

Wireframe : OFF
Filter Outliers : OFF
Display : Timing - Median
Closest Points : OFF

**Figure 9: Benchmark's results GUI for OpenCollBench, which showing generated heatmap based on benchmarking results. Left panel (red box) enable user to select different mapping value, middle panel (green box) showing generated heatmap, and right panel (blue box) showing mapping color value.**

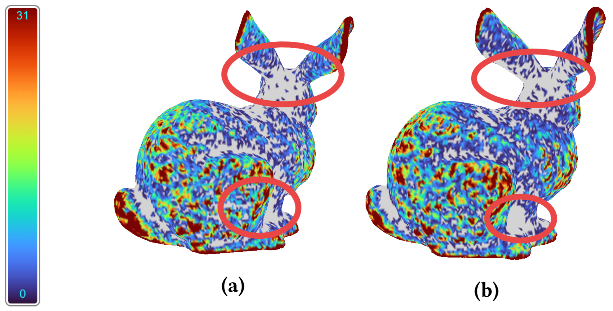


**Figure 10: Heatmaps of object bunny with around 65k polygons based on configuration's density of around 200k configuration using position finding method (a) Grid, and (b) Sphere at a relative distance of 0.0. The *Grid* method is able to generate more configurations at concave area (red circles) compared with *Sphere* method.**

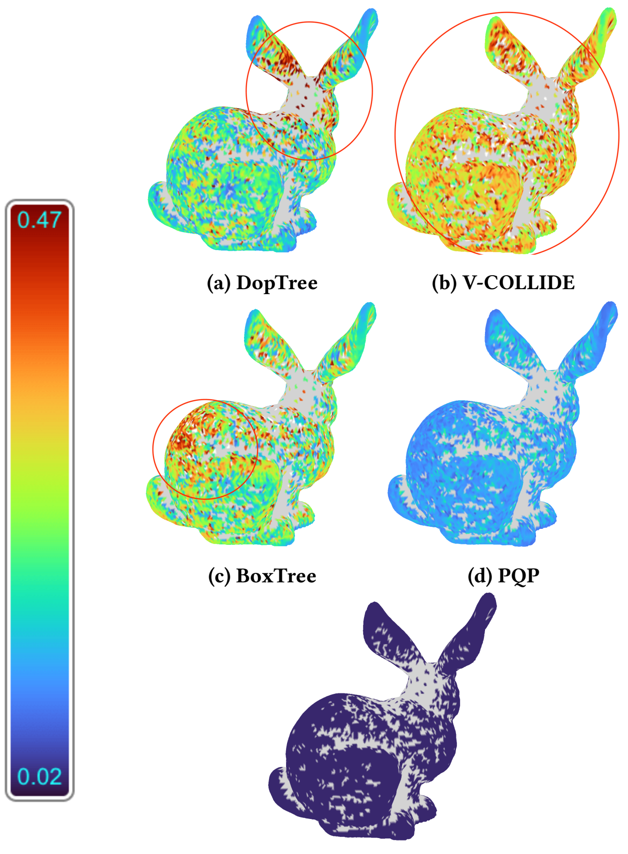
http://opencollbench.com. Currently, only files in the OBJ format can be uploaded by the user. In general, the benchmarking suite and three.js support a wide variety of different 3D file formats; however, they have to be integrated manually into three.js. For this reason, but also for security reasons, we decided, at the moment, to restrict the upload to the mostly used plain file format and add support for further file formats later on user request.

First, we have investigated the performance of our benchmarking server. In the case that the user does not choose a predefined scenario but decides to upload his own objects, he has to generate a set of configurations. According to [Trenkel et al. 2007], the user can choose between the *Grid* and *Sphere* method. For a user-definable number of configurations, the sphere method is faster, but it may fail to generate some interesting contact scenarios, especially in the case of concave objects, whereas the grid method is able to generate a wider variety of configurations but requires more computation time (see, e.g., the area around the ears and the foot in Figure 10). In general, computing configurations can be rather time-consuming; both the grid as well as the sphere method require up to 20 minutes to generate around 200k configurations for a pair of objects consisting of a total of 130k polygons (see Figure 5). In the case of close distances, the sphere method converges very quickly because it is based on a BVH distance algorithm. In case of larger distances, a lot more BV-pairs have to be considered to find the closest distance during the traversal, i.e., the pruning takes longer. We did not expect such a large difference between the individual distance as are shown by Figure 5, especially for distance 0.0, where we recognized a speed-up of more than an order of magnitude for the sphere method. However, the tendency of this behavior is independent of the object; at least it appeared with all our benchmarking scenarios. We will further investigate this in the future. While the configuration computation is relatively slow, the actual benchmarking can be done quickly. Benchmarking 200K configurations for a pair of objects with a total of 130k polygons requires only 2 minutes in case of the worst-case distance of 0.0.

In Section 3.2, we have introduced our new heatmap visualization that allows investigating the algorithm's performance on a sub-object level. In this section, we will present a few findings from this visualization. We use the google turbo colormap [Google 2019] to map different kinds of benchmarking data to the vertices. This data can be, for instance, average or median timing, the deviations of the timings, the density, e.g., a counter how often a particular polygon realizes the minimum distance between the objects for a given number of configurations or the number of BV and polygon tests. The average and minimum CD times per-vertex help us to identify regions of the object where the CD requires more time than in other regions (see Figure 3). However, in the case of large differences in the values or measurement inaccuracies, our optional outlier detection can be enabled, as described in the previous section. This allows us to find the more complicated CD configurations, that with the largest median CD times, close to the center of the pipes object as expected. Investigating the timing deviations helps us to identify regions that are susceptible to different configurations: e.g., Figure 12 shows the mapping of median absolute deviation timing for an object using the DOPTree algorithm. We can see that the performance checking the outer regions of the object is relatively independent of the configuration, whereas, for the inner regions, the configuration matters. Moreover, we can identify regions that are hardly ever colliding, independently of the colliding object's configuration. To visualize this, we map the configuration's density to vertex color. Figure 13 shows the heatmap for the extremely concave Lustre object. The inside of the object is hardly checked by algorithms. However, it also shows small regions on the extremal points of the objects that are hit very often. In the future, it could be helpful to optimize CD algorithms for exactly such high-density regions why building looser BVs for less dense regions, e.g., by stopping the BV construction earlier and thus, storing multiple polygons in a single leaf node.

We can also spread the heatmap coloring visualization through the results of several algorithms: Figure 11 shows the median CD check times for the bunny object with 65k polygon with a single unified coloring for all algorithms. It is easy to detect the fastest algorithm by the deep blue color, which is, in this example, the SimDOP. For some algorithms, we can find different critical regions; for the DopTree, checking the regions between the ears is the most time consuming (see Figure 11a), whereas the Boxtree has a bottleneck at the back of the bunny (see Figure 11c). V-COLLIDE, PQP, and SIMDop seem to perform independent of the region, at least in this unified visualization.

Beyond boolean CD, our benchmarking suite can be used to evaluate PQ algorithms. Obviously, PQ is more complicated than simple CD checks: classical BVH-based CD algorithms can prune non-overlapping parts earlier according to the *Separating Axis Theorem (SAT)*. Hence, when using the same BVH, the PQ performs worse than the CD BVH. Figure 14 shows the benchmarking result using SIMDop, an algorithm that supports both CD & PQ checks. The CD check remains fast & stable across all configurations, whereas PQ checks slow & differs between regions compared with the CD check. In the case of larger distances, the minimum distance is usually found close to the objects' extremal points, i.e., on the convex hull of the object. We can find this observation by visualizing the density with respect to the distance: Figure 15 shows heatmaps for a chair object with 113k polygons for the various relative distance between objects. The configurations were generated using the *Grid* method and have around 200k configurations each.



**(a) DopTree** **(b) V-COLLIDE**

**(c) BoxTree** **(d) PQP**

**(e) SIMDop**

**Figure 11: Heatmaps based on median value (timing in milisec) for object bunny with 65k polygons based on relative median value of various CD algorithms timings after removing outliers. The red circles show slower regions for particular algorithms.**

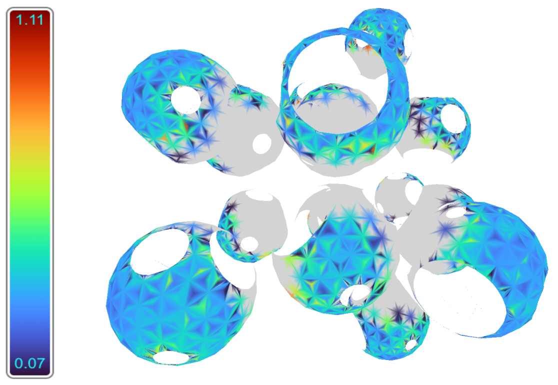


**Figure 12: Heatmap result of object schwamm with 95k polygons based on median absolute deviation (timing in milisec) using DOPTree. The outer region does not fluctuate much, whereas the inner region fluctuates up to 1.1 milisec between slightly different configurations.**

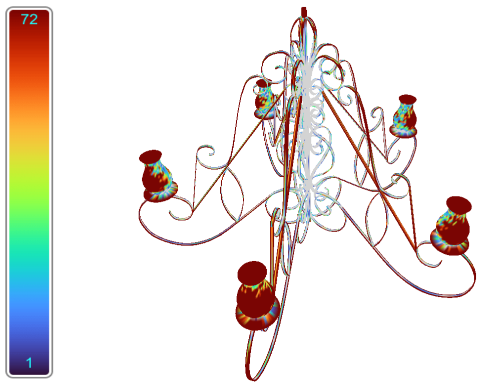
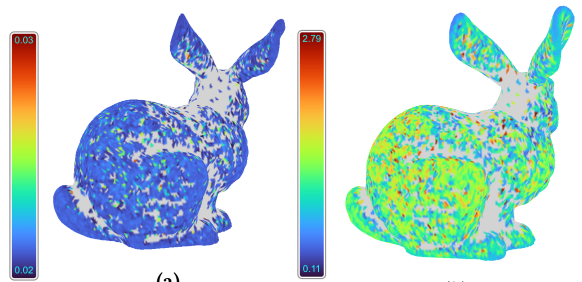
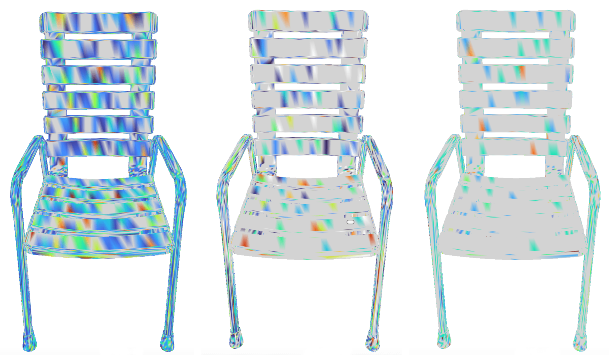


**Figure 13: Heatmap of object lustre with 120k polygons based configuration's density of 200k configurations generated using *Grid* method after removing outlier. The inner region rarely checked by algorithms, whereas the outer region heavily tested.**



**(a)** **(b)**

**Figure 14: Heatmaps of object bunny with 65k polygons based on median value (timing in milisec) of 200k configurations for (a) CD, and (b) PQ without SIMD traversal, using SIMDop algorithms at relative distance of 0.0. The CD check remains stable across configurations, whereas PQ fluctuates between regions.**



**(a)** **(b)** **(c)**

**Figure 15: Heatmaps of object chair with 70k polygons based on configuration's density generated by grid method at relative distance (a) 0.0, (b) 0.2, and (c) 0.4. The further the relative distance between objects, the fewer object regions checked by algorithms.**

## 5 CONCLUSIONS AND FUTURE WORK

We have presented OpenCollBench, a benchmarking architecture for collision detection and proximity algorithms that offers the benchmarking procedure as an open web service to the public. The goal is to make complicated and time-consuming benchmarking accessible for both expert and non-expert users. We have addressed this goal by proposing a combination of a simple yet adjustable user interface with a dedicated hardware platform that guarantees reproducible and comparable results. Additionally, we have presented an extension to a sub-object accuracy for the analysis of the benchmarking results. The idea is to use heatmaps to visualize information gathered by the benchmark. This allows the user to identify critical parts of their objects, and it enables a better understanding of the behavior and characteristics of the particular collision detection algorithm.

Our approach also offers interesting avenues for future work: for instance, currently, OpenCollBench is restricted to narrow phase collision detection and proximity queries for rigid objects that run on the CPU. Obviously, we want to extend our benchmark to cover more cases related to collision detection, like broad phase CD, deformable objects, GPU-based algorithms, other kinds of object representation than polygonal objects, to name but a few. We also plan to include real penetration scenarios, e.g., the relative penetration volume, according to [Weller et al. 2010], that can be used to compute additional configurations. In general, we want to include more collision detection libraries. In the future, we plan to offer researchers and developers an automatic upload of their libraries to the OpenCollBench framework. However, this may result in security risks, which is the main reason that currently, the inclusion of new algorithms is curated by the authors. Moreover, we want to use the information gained from the extended heatmap visualization to improve existing collision detection algorithms or even develop completely new ones. Our results already provide hints that BVH-based algorithms can be optimized by, for instance, optimizing the polygonization in parts of the objects, e.g., by transparently performing local subdivision steps or by optimizing the BVH construction. We also consider a hybrid algorithm that automatically chooses the optimal CD algorithm depending on the objects' actual configuration. This could be realized by an AI-based approach. Finally, we consider extending the idea of a benchmark as a service to other kinds of algorithms, especially in the computer graphics context: acceleration data structures for ray tracing could be a first interesting topic for this.

## ACKNOWLEDGMENTS

The research reported in this paper has been (partially) supported by the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 "EASE - Everyday Activity Science and Engineering", University of Bremen (http://www.ease-crc.org/). The research was conducted in subproject R03 <Embodied simulation-enabled reasoning>.

## REFERENCES

Stefano Caselli, Monica Reggiani, and M. Mazzoli. 2002. Exploiting Advanced Collision Detection Libraries in a Probabilistic Motion Planner.. In *WSCG*. 103–110.

Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixé. 2020. Mot20: A benchmark for multi object tracking in crowded scenes. *arXiv preprint arXiv:2003.09003* (2020).

Engin Deniz Diktas and Ali Vahit Sahiner. 2008. A benchmarking framework for static collision detection. (2008).

R. Gillard and G. A. E. Vandenbosch. 2009. SoftLAB, a European web-service for antenna software benchmark. In *2009 3rd European Conference on Antennas and Propagation*. 2736–2740.

Google. 2019. *Turbo, An Improved Rainbow Colormap for Visualization.* https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html

Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 171–180.

David Hsu, Lydia E Kavraki, Jean-Claude Latombe, Rajeev Motwani, Stephen Sorkin, et al. 1998. On finding narrow passages with probabilistic roadmap planners. In *Robotics: the algorithmic perspective: 1998 workshop on the algorithmic foundations of robotics*. 141–154.

Philip M Hubbard. 1993. Interactive collision detection. In *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*. IEEE, 24–31.

Philip M. Hubbard. 1996. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics* 15, 3 (July 1996), 179–210.

Thomas C Hudson, Ming C Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. 1997. V-COLLIDE: Accelerated collision detection for VRML. In *Proceedings of the second symposium on Virtual reality modeling language*. 117–ff.

James Thomas Klosowski. 1998. *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments.* Ph.D. Dissertation. State University of New York at Stony Brook. Adviser-Joseph S. Mitchell.

S. Krishnan, M. Gopi, M. Lin, Dinesh Manocha, and A. Pattekar. 1998. Rapid and Accurate Contact Determination between Spline Models using ShellTrees. *Computer Graphics Forum* 17, 3 (1998), 315–326.

E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. 1999a. Fast proximity queries with swept sphere volumes. In *Technical Report TR99-018*.

Eric Larsen, Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. 1999b. *Fast proximity queries with swept sphere volumes.* Technical Report. Department of Computer Science, University of North Carolina.

Jonas Lext, Ulf Assarsson, and Tomas Moller. 2001. A benchmark for animated ray tracing. *IEEE Computer Graphics and Applications* 21, 2 (2001), 22–31.

Miguel A. Otaduy and Ming C. Lin. 2003. CLODs: Dual Hierarchies for Multiresolution Collision Detection. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Aachen, Germany) *(SGP '03)*. Eurographics Association, Goslar, DEU, 94–101.

T. Tan, R. Weller, and G. Zachmann. 2019. SIMDop: SIMD optimized Bounding Volume Hierarchies for Collision Detection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 7256–7263.

Sven Trenkel, René Weller, and Gabriel Zachmann. 2007. A Benchmarking Suite for Static Collision Detection Algorithms. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, Václav Skala (Ed.). Union Agency, Plzen, Czech Republic. http://cg.in.tu-clausthal.de/research/colldet_benchmark

Gino van den Bergen. 1998. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools* 2, 4 (Jan. 1998), 1–13. http://dl.acm.org/citation.cfm?id=763345.763346

Rene Weller, Mikel Sagardia, David Mainzer, Thomas Hulin, Gabriel Zachmann, and Carsten Preusche. 2010. A benchmarking suite for 6-dof real time collision response algorithms. In *Proceedings of the 17th ACM symposium on virtual reality software and technology*. 63–70.

J-L Widlowski, M Robustelli, M Disney, J-P Gastellu-Etchegorry, T Lavergne, P Lewis, PRJ North, B Pinty, R Thompson, and MM Verstraete. 2008. The RAMI On-line Model Checker (ROMC): A web-based benchmarking facility for canopy reflectance models. *Remote Sensing of Environment* 112, 3 (2008), 1144–1150.

Muiris Woulfe and Michael Manzke. 2009. A framework for benchmarking interactive collision detection. In *Proceedings of the 25th Spring Conference on Computer Graphics*. 205–212.

Gabriel Zachmann. 1995. The BoxTree: Exact and Fast Collision Detection of Arbitrary Polyhedra. In *SIVE Workshop*. 104–112.

Gabriel Zachmann. 1998. Rapid Collision Detection by Dynamically Aligned DOP-Trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*. Atlanta, Georgia, 90–97.