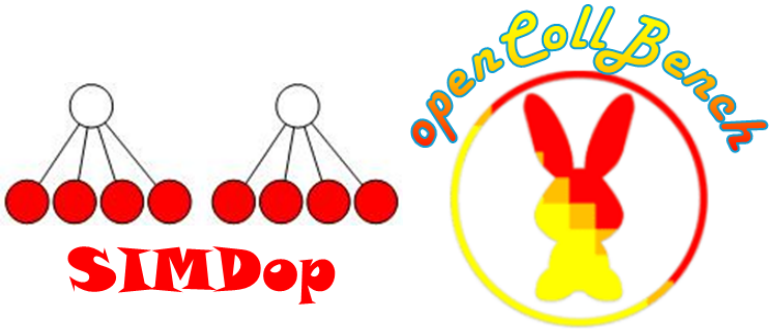
GEOMETRIC COMPUTING FOR SIMULATION-BASED ROBOT PLANNING



... and more

Dem Fachbereich Informatik
der Universität Bremen
eingereichte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Toni Tan, M.Sc

Referenten der Arbeit:	Prof. Dr. Gabriel Zachmann
	Prof. Michael Manzke PhD
Tag der Einreichung:	April 04, 2024
Tag des Kolloqiums:	July 03, 2024

Toni Tan: *Geometric Computing for Simulation-Based Robot Planning,*

To both my children, Rayleigh and Raelyn:
Without whose interruptions, detours, and constant questions, this thesis would have been finished years earlier.

DECLARATION

I hereby declare that I am the legitimate author of this Dissertation and that it is my original work. No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education. I hold the University of Bremen harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

Bremen, April 04, 2024

Toni Tan

ABSTRACT

Simulation-based robot planning is a popular approach in robotics that involves using computer simulations to plan and optimize robot motions by envisioning the outcome of generated plans before their execution in the real world. This approach offers several benefits, including the ability to evaluate multiple motion plans, reduce trial-and-error in physical experimentation, and enhance safety by identifying potential collisions and other hazards before executing a motion.

Although this approach can significantly benefit robotic manipulation tasks, such simulations are still computationally expensive and may require more computing power than the robotic agents can provide. In addition, uncertainties arising from, i.e., perception or simulation models must be taken into account. Current approaches often require running simulations multiple times with varying parameters to account for these uncertainties, making real-time action planning and execution difficult.

This thesis presents an accelerated geometric computation, i. e., Collision Detection (CD) methods for such simulation, precisely an algorithm based on Bounding Volume Hierarchies (BVHs) and Single Instruction Multiple Data (SIMD) instruction sets. The main idea is to increase the branching factor of Bounding Volume Hierarchy (BVH) according to available SIMD width and simultaneously test Bounding Volume (BV) nodes for intersection in parallel. In addition, this thesis presents compression strategies for BVH-based CD implemented on two existing CD algorithms, namely Doptree and Boxtree. The idea is to remove redundant information from BVHs, and compress 32-bit floating points used to represent BVHs. This greatly increases the number of simultaneous simulations done in parallel by robotic agents, with most benefitting remote robots, as their computing power is often limited.

Furthermore, this thesis presents an idea of benchmarking as an online service. In the literature, it is quite often that the results of proposed algorithms are difficult to replicate due to missing hardware/-software and different computing configurations. Combined with the idea of using a virtual machine to safely execute user-uploaded algorithms makes it possible to safely run benchmarks as an online service. Not only are the results reproducible, but they are also comparable, as they are done within the same hardware/software configurations.

Finally, this thesis investigates an idea to address uncertainties by incorporating them into simulations. The main concept is to integrate uncertainty as a a probability distribution into CD algorithms. In this sense, CD algorithms will not only report collisions but also the

probability when a collision occurs. The outcome is not a simple final state of the simulation but rather a probability map reflecting a continuous distribution of final states.

ZUSAMMENFASSUNG

Die simulationsbasierte Roboterplanung ist ein beliebter Ansatz in der Robotik, der Computer-Simulationen nutzt, um Roboterbewegungen zu planen und zu optimieren, indem das Ergebnis generierter Pläne vor ihrer Ausführung in der realen Welt vorhergesagt wird. Dieser Ansatz bietet mehrere Vorteile, darunter die Möglichkeit, mehrere Bewegungspläne zu bewerten, Versuchs-und-Irrtum-Methoden bei physischen Experimenten zu reduzieren und die Sicherheit zu erhöhen, indem potenzielle Kollisionen und andere Gefahren vor der Ausführung einer Bewegung identifiziert werden. Obwohl dieser Ansatz für robotische Manipulationen von großem Nutzen sein kann, sind solche Simulationen immer noch rechenintensiv und erfordern möglicherweise mehr Rechenleistung, als die Roboter bereitstellen können. Darüber hinaus müssen Unsicherheiten, die aus der Wahrnehmung oder den Simulationsmodellen resultieren, berücksichtigt werden. Aktuelle Ansätze erfordern oft das mehrfache Ausführen von Simulationen mit variierenden Parametern, um diese Unsicherheiten zu berücksichtigen, was eine Echtzeit-Aktionsplanung und -ausführung erschwert.

Diese Arbeit präsentiert eine beschleunigte geometrische Berechnung, d.h. Methode zur Kollisionserkennung für solche Simulationen, genau einen Algorithmus basierend auf BVHs und SIMD Befehlssätzen. Die Hauptidee besteht darin, den Verzweigungsgrad der BVH entsprechend der verfügbaren SIMD-Breite zu erhöhen und gleichzeitig BV-Knoten parallel auf Schnitt zu testen. Darüber hinaus präsentiert diese Arbeit Kompressionsstrategien für BVH-basierte Kollisionserkennung, die auf zwei bestehenden algorithmen, nämlich Doptree und Boxtree, implementiert sind. Die Idee besteht darin, redundante Informationen aus BVHs zu entfernen und 32-Bit-Fließkommazahlen zu komprimieren, die zur Darstellung von BVHs verwendet werden. Dies erhöht die Anzahl gleichzeitiger Simulationen, die von Robotern parallel durchgeführt werden, wobei besonders entfernte Roboter profitieren, da ihre Rechenleistung oft begrenzt ist.

Darüber hinaus präsentiert diese Arbeit die Idee eines Benchmarkings als Online-Dienst. In der Literatur ist es häufig so, dass die Ergebnisse vorgeschlagener Algorithmen aufgrund fehlender Hardware-/Software- und unterschiedlicher Rechnerkonfigurationen schwer zu reproduzieren sind. In Kombination mit der Idee, eine virtuelle Maschine zu verwenden, um benutzerhochgeladene Algorithmen sicher auszuführen, ist es möglich, Benchmarks sicher als Online-Dienst durchzuführen. Nicht nur sind die Ergebnisse reproduzierbar, sondern sie sind auch vergleichbar, da sie innerhalb derselben Hardware-/Software-Konfigurationen durchgeführt werden.

Schließlich untersucht diese Arbeit eine Idee, Unsicherheiten anzugehen, indem sie diese in Simulationen integriert. Das Hauptkonzept besteht darin, Unsicherheit als Wahrscheinlichkeitsverteilung in CD-Algorithmen zu integrieren. In diesem Sinne werden CD-Algorithmen nicht nur Kollisionen melden, sondern auch die Wahrscheinlichkeit angeben, wann eine Kollision auftritt. Das Ergebnis ist nicht nur ein einfacher Endzustand der Simulation, sondern vielmehr eine Wahrscheinlichkeitskarte, die eine kontinuierliche Verteilung von Endzuständen widerspiegelt.

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to all those who have supported and encouraged me throughout my dissertation journey.

Firstly, I would like to thank my supervisor Prof. Dr. Gabriel Zachmann and Prof. Michael Manzke PhD for their invaluable guidance, patience, and expertise. Without their support and feedback, this dissertation would not have been possible.

Moreover, I wish to thank all my co-authors for the fruitful collaboration without which this thesis undoubtedly would not exist. Namely, Dr. René Weller, Janis Rosskamp, Franklin Kenghagho Kenfack, and Michael Neumann.

I would also like to extend my gratitude to the faculty and staff at Computer Graphics and Virtual Reality Research Lab (CGVR) at the University of Bremen, who provided me with the necessary resources, opportunities, and a conducive learning environment. In alphabetical order: Abhishek Srinivas, Andre Mühlenbrock, Christoph Schröder-Dering, Hermann Meißenhelter, Helga Reinermann, Jörn Teuber, Maximilian Kaluschke, Philipp Dittmann, Dr. Roland Fischer, Sabine Dolhs, Tanja Rethemeyer, Thomas Hudcovic, and Dr. Xizhi Li.

I am also indebted to my family and friends who stood by me during this challenging phase of my life. Their unwavering support, motivation, and love have been a source of inspiration to me.

Last but not least, I would like to express gratitude for the contribution of my students who have been a source of intellectual stimulation and motivation throughout my journey. In alphabetical order: Alexander Schwochow, Anjishnu Mukherjee, Issawat Nilavongse, Jan-Eric Oltmanns, Paul Duhr, Torben Groß, and Waralee Tanaphantaruk.

Thank you all for your support, encouragement, and patience.

CONTENTS

ACRONYMS

AABB Axis Aligned Bounding Box

ACM Association for Computing Machinery

AVX Advanced Vector Extensions

AVX-512 512-bit Advanced Vector Extensions

BNG Batch Neural Gas

BV Bounding Volume

BVH Bounding Volume Hierarchy

BVHs Bounding Volume Hierarchies

BOS Benchmark as Online Service

CD Collision Detection

CCD Continuous Collision Detection

CPU Central Processing Unit

DoF Degrees of Freedom

DOP Discrete Oriented Polytope

DLL Dynamic-Link Library

GPU Graphics Processing Unit

GUI Graphical User Interface

HRI Human-Robot Interaction

IST Inner Sphere Tree

k-DOP k - Discrete Oriented Polytope

mRBPF marginal Rao-Blackwellized Particle Filter

NEEM Narrative-Enabled Episodic Memory

NG Neural Gas

OBB Oriented Bounding Box

OS Operating System

PQ Proximity Query

PQP Proximity Query Package

PRM Probabilistic Roadmap

RAM Random Access Memory

RCE Remote Code Execution

RRT Rapidly-Exploring Random Trees

SAH Surface Area Heuristic

SIMD Single Instruction Multiple Data

SIGGRAPH Special Interest Group on Computer Graphics and Interactive Techniques

SSE Streaming SIMD Extensions

VC V-COLLIDE

VM Virtual Machine

VR Virtual Reality

1 INTRODUCTION

In recent years, there have been notable advancements in the field of robot planning. These advances have introduced new algorithms and techniques with the goal of creating efficient and effective paths for robots to navigate within their environments. One specific area of interest in this field involves the use of mental simulation as a key component in simulation-based methods for generating motion plans.

Mental simulation, a core concept in cognitive science (Battaglia, Hamrick, and Tenenbaum, 2013; Hesslow, 2012), has played a critical role in enabling robots to engage in realistic, robust, and efficient reasoning about their intended actions. This reasoning process relies on the use of mental simulations, which employ qualitative reasoning techniques to think about actions and their expected outcomes. This approach involves translating the observed scenes and planned actions into parametrized simulation problems. Multiple detailed physics-based simulations are then sampled to simulate the robot's action plan, capturing the evolution of states within appropriate data structures. Subsequently, these sub-symbolic data structures are transformed into interval-based first-order symbolic representations known as Narrative-Enabled Episodic Memory (NEEM). NEEM serves as qualitative representations, providing insights into how the robot's actions evolve.

For robots to excel in everyday tasks, they must possess the ability to think flexibly about the actions they intend to carry out. For example, when faced with the task of picking up a stack of plates and cups, a robot should be able to quickly assess the likelihood of objects toppling over and make appropriate decisions accordingly. Similarly, the robot should be able to anticipate whether to use one hand or both when lifting a pile of plates. Moreover, when handling an open ketchup bottle, the robot should be able to predict the potential consequences of gripping the bottle too tightly. However, the challenge of performing such inferences realistically and promptly remains open.

Recent years have also seen significant progress in physically-based simulation and animation, leading to the creation of increasingly realistic representations of human motions and the interactions between physical objects (as shown in Figure 1.1). Physics engines such as PhysX, Havok, or Bullet, accelerated by Graphics Processing Unit (GPU), have played a pivotal role in these advancements. Additionally, strides have been made in the development of animation algorithms. These developments provide valuable resources and open up opportunities for research in robotics and artificial intelligence

to explore prediction and envisioning algorithms that were previously considered unrealistic but can now be pursued in a much more realistic setting.

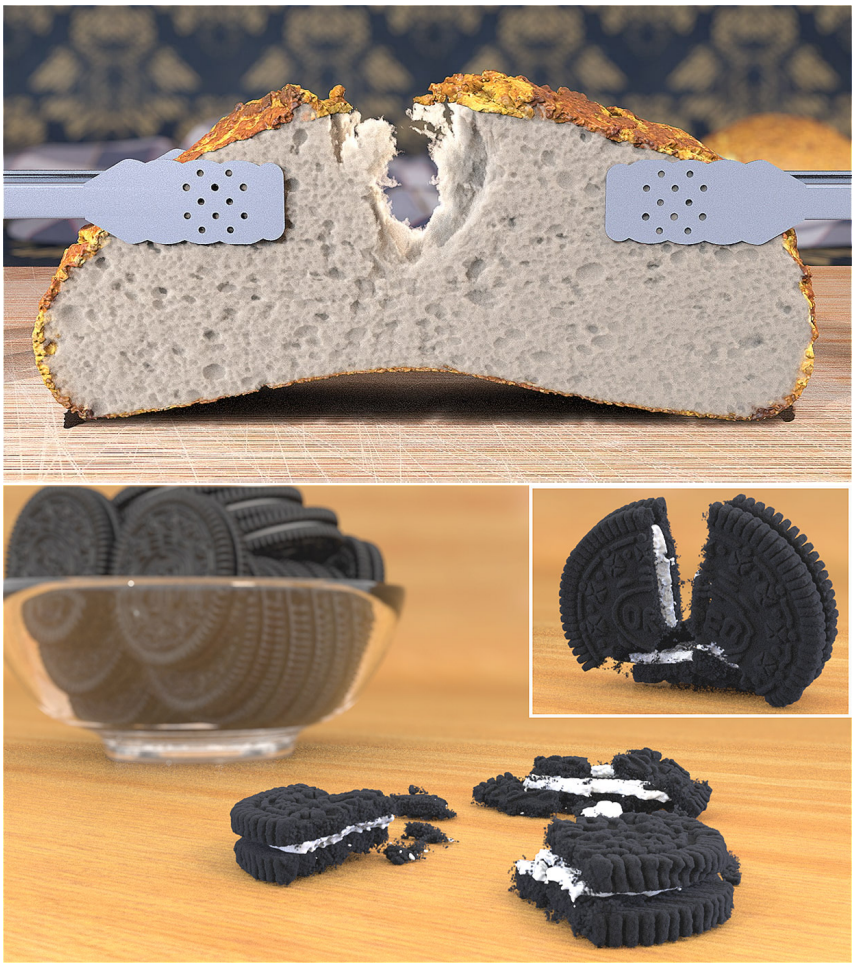


Figure 1.1: Recent advancements in physics-based simulations have led to significant progress in accurately modeling the behavior of materials, resulting in more realistic simulations. (Wolper et al., 2019)

Still, using simulation and animation tech for predicting actions of robots based on physics faces some extra challenges. These challenges include:

- **Partial observability**: Robotic agents often encounter situations where they lack complete information about the physical world being simulated, resulting in difficulties in deriving a comprehensive dynamic model (Richter-Klug et al., 2022).
- **Uncertainty arising from sensing and action noise**: Estimations of physical parameters are frequently subject to inaccuracies due to uncertainties stemming from sensing processes. Consequently, the parametrizations of simulations may become less accurate.

- **Semantic and qualitative events**: Robotic agents typically prioritize the detection of specific events and their corresponding effects rather than attaining precise, quantitative physical state estimates. Consequently, simulation methods need to be extended to detect and capture such events accurately (Haidu and Beetz, 2021).

- **Super-realtime and ensemble simulations**: Simulation methods need to operate at significantly faster-than-realtime speeds to swiftly conduct an ensemble of similar simulated experiments. This capability allows robotic agents to perform multiple envisionings rapidly. Furthermore, simulation outputs must provide more comprehensive information beyond just the final configuration, as is commonly required in manual tuning and control of animations.

To bridge these gaps, this dissertation focuses on investigating and developing faster-than-realtime simulation methods. By advancing the current state-of-the-art in simulation techniques, the aim is to enable robotic agents to perform simulations at accelerated speeds, surpassing real-time requirements. This allows for the rapid execution of multiple simulated experiments, enabling the robotic agents to perform numerous envisionings and obtain valuable insights and predictions in a timely manner.

Additionally, this dissertation also investigates the uncertainty within simulation. With a focus on addressing uncertainties arising from sensing and action noise, the research explores techniques and methodologies to effectively model, quantify, and manage uncertainty within the simulation framework. By incorporating uncertainty into the simulation process, more accurate and reliable predictions of action effects can be obtained, enhancing the overall robustness and adaptability of robotic agents in real-world scenarios.

Through these investigations, this dissertation aims to contribute to the advancement of physically-based simulation and animation technologies, providing valuable insights and solutions to the challenges of super-realtime and ensemble simulations, as well as uncertainties arising from sensing and action noise. Ultimately, this research endeavors to enhance the capabilities of robotic agents in reasoning, planning, and executing actions in dynamic and uncertain environments.

- **Research Question 1:** The CD is the computational bottleneck in most sampling-based algorithms, that requires up to 90% of computation time (Reggiani, Mazzoli, and Caselli, 2002). The majority of CD algorithms are BVHs-based with branching factor of 2 (binary). According to Zachmann and Langetepe (2003), the optimum branching factor can be larger. Additionally, the parallelization of the simultaneous traversal for CD is not obvious. Actually, due to their recursive nature, it is not very well

suited for massively parallel acceleration on the GPU (especially BVHs-based). Furthermore, particularly in the context of online planning, powerful GPU is typically unavailable.

Therefore, Research Question 1 investigates: How to find optimal branching factor for CD algorithms and employ it for parallelization suitable for online planning?

- **Research Question 2:** CD algorithms are influenced by various factors, including object shape, relative size, distance between objects, and distributions of geometric primitives. However, these factors are often not thoroughly discussed in existing algorithm proposals. Benchmarking in CD is typically performed using self-defined objects and scenarios, which may not provide a fair evaluation of existing algorithms. This approach not only makes it challenging to effectively compare different algorithms, but it also presents technical difficulties, such as the availability of existing algorithms over time.

  Additionally, the large number of parameters involved and the integration of existing CD algorithms contribute to the complexity and time-consuming nature of benchmarking. As a result, there is an insufficient understanding of why a particular algorithm performs better or worse in a given scenario. Even minor changes in transformations or object properties, such as a slightly modified polygonization of an object, can lead to significantly different results.

  Therefore, Research Question 2 aims to investigate how to design a benchmark that enables fair and reproducible comparisons of CD algorithms. The goal is to address the challenges of parameter selection, scenario design, and result interpretation to provide meaningful insights into algorithm performance. By developing a comparable and reproducible benchmark, researchers can gain a better understanding of the strengths and weaknesses of different algorithms and identify the factors that contribute to their performance variations.

- **Research Question 3:** The current state of physically-based simulation development is well-known to produce plausible effects for physics simulation. Although the result is convincing, simulation is subject to uncertainty. In a general sense, uncertainty can arise from three sources, lack of knowledge about the system being simulated *(simulation uncertainty)*, uncertainty in the input data (*input uncertainty*), and, the choice of model and assumptions (*structural uncertainty*) (McKay, Morrison, and Upton, 1999). All of these can lead to errors in the results of the simulation.

  Therefore, Research Question 3 investigates: how to simulate by drawing samples from probabilistic distributions over physics

parameters? The outcome is not a simple final state of the simulation but rather a probability map reflecting a continuous distribution of final states.

1.1 THESIS OUTLINE

This thesis is organized as follows: In Chapter 2, we provide an overview of simulation-based robot planning, focusing on the challenges involved. In Chapter 3, we introduce an optimized method for improving the performance of BVH-based CD, demonstrating our method's ability to meet real-time requirements in simulation-based robot planning. In Chapter 4, we present techniques for reducing memory footprint for BVH, allowing robot agents to perform more computations simultaneously within their computational limits, which is crucial for online robots.

Moving on to Chapter 5, we explore the concept of online benchmarking to ensure reproducible and comparable benchmark results. By adopting this approach, we aim to enable fair comparisons and improve our understanding of existing algorithms for CD and Proximity Query (PQ).

In Chapter 6, we propose an innovative method to address uncertainty in simulation-based motion planning. We suggest integrating continuous probability distributions into CD algorithms to handle uncertainty effectively. This new approach reduces the number of simulations needed to obtain reliable results significantly. Additionally, Chapter 7 presents applications developed using CD algorithms. Finally, in Chapter 8, we conclude this dissertation by discussing the encountered limitations during the research and outlining future directions for further investigation.

2

USE OF SIMULATION FOR ROBOT PLANNING

Robots have evolved beyond their rigid existence on assembly lines, where they were once limited to following scripted instructions. Thanks to advancements in artificial intelligence, the next generation of robots will possess enhanced mobility and decision-making abilities. These robots will be flexible, adaptable, and capable of interacting with humans, operating in dynamic and unpredictable environments. They will assist social workers in nursing homes (Huisman and Kort, 2019; Kyrarini et al., 2021), tutor students in schools (Konijn et al., 2022; Pasalidou, Fachantidis, and Koiou, 2023), autonomous shelf refilling (Cavallo et al., 2022), detect gas and oil leaks underwater (Hu et al., 2022; Wu et al., 2023), take part in search-and-rescue missions (Han et al., 2022; Lindqvist et al., 2022), and even assist in surgical procedures (Püschel, Schafmayer, and Groß, 2022; Zhao et al., 2022), even in challenging and cluttered environments.

Creating a new robot involves two slow parts: the engineering design and the control policy design. Even though it's crucial to test these robots in the real world, simulation can speed up and make the engineering design process cheaper. Simulations let us thoroughly test and improve what robots can do, paving the way for new uses and building safer, more effective robotic systems.

In control policy design, there's a special focus on using mental simulation in simulation-based methods to come up with motion plans. This helps robots imagine and think through the actions they want to take.

Additionally, simulation offers various benefits and opportunities, including:

- *Generating training data:* Simulation allows for the generation of large amounts of training data quickly and cost-effectively. Machine learning algorithms often require extensive data for training, and simulations provide an ideal platform to generate diverse scenarios and annotations for training robots. This enables systems to learn from their mistakes and improve their performance (Mania and Beetz, 2019).

- *Facilitating understanding of Human-Robot Interaction (HRI):* HRI is a crucial aspect of robotics (Fong, Nourbakhsh, and Dautenhahn, 2003). Simulation tools can be developed to better represent the psycho-social nature of HRI, enabling the study of human-robot collaboration and decision-making processes. By creating simulated environments, researchers can explore different scenarios,

analyze the impact of various factors on trust and collaboration, and establish a baseline for more effective human-robot interaction (Metcalfe et al., 2017).

- *Accelerating engineering design and reducing costs:* Simulation significantly speeds up the engineering design cycle. Instead of physically testing each iteration, simulations allow for quick and parallel evaluation of candidate solutions (Chinesta et al., 2020). This reduces time, costs, and potential risks associated with physical testing. Additionally, simulations enable testing in environments that are impractical or inaccessible for real-world testing, such as extreme conditions or remote locations.

- *Safe and controlled virtual testing:* Simulations provide a safe and controlled environment for testing robots. In complex scenarios involving multiple robots or human-robot interactions, designing and verifying these systems can be challenging and time-consuming in the physical world. Simulation allows for systematic testing and evaluation of multi-robot systems, facilitating the design of collaborative behaviors, coordination strategies, and decision-making algorithms (Burgard et al., 2005; Tan and Zheng, 2013). It also enables the exploration of scenarios that are difficult to replicate physically, such as environmental monitoring, surveillance, or infrastructure management.

2.1 SIMULATION-BASED INTERNAL MODELS FOR ROBOTS

Internal models in robotics can be classified into two main types: forward models (predictors) and inverse models (controllers), supplemented by models that predict physical properties of the environment. Forward models anticipate future sensory inputs based on motor inputs, while inverse models generate motor commands to achieve desired sensory inputs. These models enable the generation and testing of hypotheses about the consequences of future actions and facilitate the recognition of other agents' behavior. Anticipation can enhance navigation performance in complex tasks, with the quality of predictions being a crucial factor (Johansson and Balkenius, 2006). Additionally, the optimal amount of time predicted into the future strikes a balance between insufficient and excessive prediction, affecting performance improvement (Johansson and Balkenius, 2008). Achieving a balance in prediction capabilities empowers robots to effectively plan and adapt their actions, thereby enhancing their overall performance and efficiency across a range of tasks.

With the steady increase in computational power, there is growing interest in using actual simulators instead of learned models in robotics. For instance, Bongard, Zykov, and Lipson (2006) demonstrated a four-legged robot that utilized an internal simulation to self-model and

generate actions for locomotion. Additionally, Millard, Timmis, and Winfield (2014a,b) investigated the application of simulation-based internal models for detecting faults in swarm robotics systems. These studies showcase the potential of using simulation-based internal models to enhance robot capabilities in tasks such as self-modeling, locomotion, and fault detection.

The use of simulation-based internal models in robots is also regarded as a promising approach for enhancing safety in highly dynamic environments. Blum, Winfield, and Hafner (2018) propose a concept where a robot incorporates a simulation of itself, other dynamic actors, and its environment within its own system. This internal model operates in real-time and enables the robot to anticipate and predict the outcomes of its own actions as well as the actions of other actors in its surroundings. Consequently, the robot can dynamically adjust its actions to actively ensure its own safety while simultaneously achieving its intended goals.

Notable robotic simulation platforms, such as Gazebo (Koenig and Howard, 2004), Webots (Michel, 2004), Player-Stage (Vaughan and Gerkey, 2007), and Morse (Echeverria et al., 2011), provide valuable environments for evaluating and optimizing robot systems, but the computationally expensive nature of simulation-based based, making a fully embodied implementation challenging.

2.2 MOTION PLANNING

Motion planning is a crucial aspect of robotics, involving the determination of a feasible path or trajectory for a robotic agent to transition from its current position to a predefined desired goal position This process necessitates the ability to circumvent obstacles and adhere to various operational constraints. The robot's physical attributes, such as its size, shape, and the placement of obstacles within its environment, are typically specified within either a two-dimensional (2D) or three-dimensional (3D) workspace. In contrast, when considering the robot's movement, it is useful to conceptualize it as a path within a configuration space. This configuration space might encompass additional dimensions beyond the spatial dimensions of the workspace, allowing for a more comprehensive representation of the robot's possible configurations and movements.

Key components in motion planning include:

- *Configuration space*, encapsulates the entirety of conceivable robot poses. In essence, it is a mathematical abstraction representing all possible configurations the robot can assume. The primary objective of motion planning is to identify a continuous and obstacle-free path within this configuration space, connecting the robot's initial configuration to its desired goal configuration.

- *Free space*, constitutes the subset of configurations within the configuration space where the robot can operate without encountering collisions with obstacles. Identifying and mapping out this free space is critical for the successful execution of motion planning algorithms.
- *Target space*, a distinct subset of the configuration space that defines the permissible locations for the robot's end-effector or target position. It specifies the region where the robot should ultimately reach, and the motion planning process aims to determine a suitable path to navigate the robot to this target location while avoiding obstacles.
- *Obstacle space*, characterizes the spatial distribution of obstacles within the workspace. This information is crucial for motion planning as it defines the regions within the configuration space that the robot must avoid to prevent collisions. Effectively representing and managing obstacle information is fundamental to ensuring safe and efficient robotic navigation.

2.2.1 *Low Dimension Spaces*

In scenarios with lower dimensions, like those encountered by self-driving cars in 2D or 3D environments, grid-based algorithms are commonly used. These algorithms involve overlaying a grid onto the configuration space, facilitating efficient path planning and collision detection.

However, in situations where a more detailed understanding of the environment is necessary, geometric algorithms come into play. These algorithms are tailored to compute the shape and connectivity of the free configuration space. Notable examples include the Visibility graph (Lubiw, Snoeyink, and Vosoughpour, 2017; Niu et al., 2019), extensively studied and applied in contexts such as coverage planning. Other techniques like Cell decomposition (Choset, 2000), Minkowski sum, and farthest ray tracing offer valuable insights into the geometry and topology of free configuration space, enabling more sophisticated and precise path planning and analysis. Geometric algorithms prove particularly useful when the complexity of the environment calls for a more nuanced approach beyond grid-based methods.

2.2.1.1 *Grid-based search*

In grid-based search algorithms, the grid overlay on the configuration space transforms it into a graph, as shown in Figure 2.1. The primary goal of this representation is to facilitate efficient path planning. Planners typically employ graph search algorithms, with A^* being a popular choice, as introduced by Hart, Nilsson, and Raphael (1968).

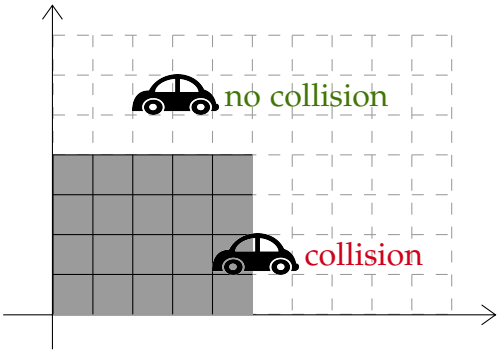


Figure 2.1: In low-dimensional environments, grid-based algorithms overlay a grid onto the configuration space, which enables efficient collision detection and path planning.

What makes A^* stand out from methods like Dijkstra is its use of heuristics to estimate the distance from the current state to the goal, enhancing exploration. The search algorithm ultimately provides the optimal path to guide the robot to its destination.

However, to ensure the final path is collision-free and safe, planners must frequently perform CD operations, as depicted in Figure 2.1. CD involves checking if, in a given state, the robot would collide with any obstacles in the environment. Notably, CD often becomes a significant performance bottleneck in these algorithms. Recent studies, like Bakhshalipour, Likhachev, and Gibbons (2022), have revealed that over 65% of the entire execution time is consumed by CD operations, underscoring its critical role and the need for optimization in this area.

2.2.2 *High Dimension Spaces*

Navigating through high-dimensional configuration spaces, such as those found in stationary robotic arm manipulators with multiple Degrees of Freedom (DoF), poses a significant and time-consuming challenge in the field of robotics. The dimension of the configuration space corresponds to the number of DoF, making it impractical to represent the entire space graphically. This challenge has prompted focused attention and solutions across various levels, encompassing algorithmic approaches and architectural innovations (Lian et al., 2018; Murray et al., 2016, 2019).

In managing these high-dimensional spaces, commonly employed algorithms for path planning include Rapidly-Exploring Random Trees (RRT) and Probabilistic Roadmap (PRM).

2.2.2.1 *Static Environment*

In static environments, PRM planning, such as those by Baek et al. (2018), Chen et al. (2021), Kavraki et al. (1996), and Santiago et al. (2017), emerges as a fundamental algorithm for navigating high-

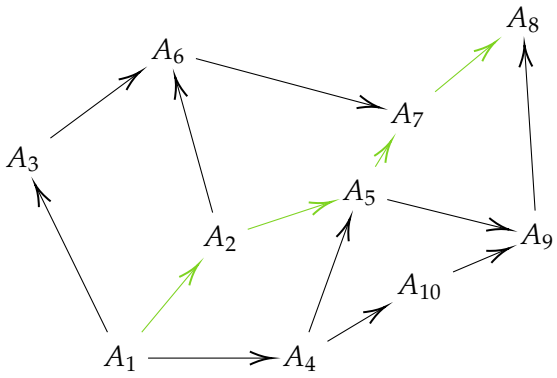


Figure 2.2: Probabilistic Road Map (PRM) algorithms draw random samples from the configuration space, evaluate them for collision-free properties, connect them to form a graph, and then utilize search algorithms to find a path from the start to the goal.

dimensional configuration spaces. PRM operates in two distinct phases: offline and online. In the offline phase, random samples are drawn from the robot's configuration space, assessed for collision-free properties, and connected to form a graph. The online phase introduces the start and goal configurations to this graph and employs search algorithms such as A^* to uncover a path from the start to the goal. The resulting path, illustrated as the green trajectory in Figure 2.2, represents the robot's route.

2.2.2.2 *Dynamic Environment*

Utilizing RRT (Cao et al., 2019; Denny et al., 2020; LaValle, 1998) in dynamic environments provides a significant advantage over PRM, especially in scenarios with frequent environmental changes. The adaptability of RRT to evolving surroundings is crucial for overcoming the limitations associated with static graph-based methods.

In dynamic environments where layouts may change unpredictably, PRM faces challenges due to its dependence on a predefined graph that quickly becomes outdated. In contrast, RRT excels in such scenarios. Dynamically extending its tree from the start to the goal configuration through random sampling allows RRT to effectively accommodate changes in the environment.

The real-time approach of randomly sampling and checking collision status enables RRT to respond promptly to alterations in the surroundings. This capability proves particularly valuable in applications where the robot or system needs to navigate through spaces with varying obstacles or configurations.

The introduction of RRT* adds an optimization layer to the path planning process by aiming for asymptotically optimal paths. This is achieved through the rewiring of the tree, adjusting connections to reduce the overall path cost. While RRT* does generate shorter paths on average, its computational demands, significantly higher than those

of RRT, pose challenges in terms of execution time (Bakhshalipour, Likhachev, and Gibbons, 2022).

The challenges associated with high collision detection and nearest neighbor search latency in both RRT* and RRT emphasize the importance of addressing these issues for practical implementation. The substantial portion of execution time dedicated to frequent rewiring operations in RRT* emphasizes the need for optimizations in this area.

2.2.3 *Computational Bottleneck*

CD indeed poses a common challenge in both PRM and RRT, and recent study (Bakhshalipour, Likhachev, and Gibbons, 2022), underscore its significant impact on overall execution time. With up to 65% of the total execution time dedicated to CD operations, the imperative need for optimization in this domain becomes apparent. As highlighted by Reggiani, Mazzoli, and Caselli (2002), collision computations can consume up to 90% of the overall computation time in sampling-based algorithms.

The decision between approximate and exact collision detection methods is crucial and depends on the complexity of the environment and the specific requirements of the application. Approximate CD offers faster computation by approximating the intersection between the robot's path and obstacles, albeit at the potential cost of accuracy in certain scenarios. On the contrary, exact CD provides precise calculations but incurs a higher computational cost.

Algorithm 1: BVHtraversal(BV a, BV b)

```
if a and b are both leaves then
    checkPrimitives(a, b)
else if a is leaf then
    forall children b_i of b do
        if a and b_i intersect then
            BVHtraversal(a, b_i)
else if b is leaf then
    forall children a_i of a do
        if a_i and b intersect then
            BVHtraversal(a_i, b)
else
    forall children a_i of a and b_i of b do
        if a_i and b_i intersect then
            BVHtraversal(a_i, b_i)
```

For algorithms working with polyhedral models, the use of BVHs for exact collision detection emerges as a popular and effective strategy. The hierarchical structure of BVHs, employing bounding volumes such as spheres, Axis Aligned Bounding Box (AABB), Oriented Bounding

Box (OBB), or k - Discrete Oriented Polytope (k-DOP), facilitates accelerated intersection queries. In this structure, a large bounding volume at the root encompasses all geometric primitives, and the primitives themselves serve as leaves in the BVH. Traversal involves recursively exploring the children of BVHs when their bounding volumes intersect, as illustrated in algorithm 1 and Figure 2.3.

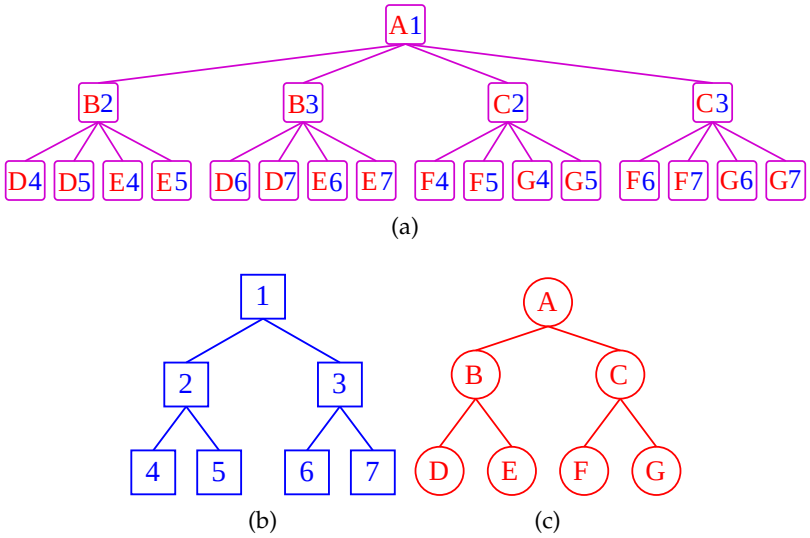


Figure 2.3: The simultaneous recursive traversal of two binary BVHs during the collision check results in a bounding volume test tree.

2.3 ENVISIONING OUTCOME OF GENERATED PLANS

In their work, Kunze and Beetz (2017) propose a framework enabling robots to anticipate the physical consequences of their intended manipulation actions through simulation-based projections. This framework empowers a robot to predict potential outcomes when executing a task in a specific manner.

The process begins by translating a qualitative physics problem into a parameterized simulation problem. A detailed physics-based simulation of the robot's plan is then conducted, with the state evolution logged into appropriate data structures. Subsequently, these sub-symbolic data structures are translated into interval-based first-order symbolic, qualitative representations known as timelines. The outcome of this envisioning process is a collection of detailed narratives represented by timelines, which are subsequently utilized to deduce answers to qualitative reasoning problems. By envisioning the potential outcomes before execution, a robot can engage in reasoning about physical phenomena, thereby avoiding undesirable situations. This approach allows robots to execute manipulation tasks with enhanced efficiency, robustness, and flexibility, even enabling them to successfully adapt to previously unknown variations of tasks.

2.3.1 *Uncertainty in Simulation*

Despite the advantages of physics simulations, they are inherently susceptible to uncertainty stemming from three primary sources: *simulation uncertainty*, arising from a lack of knowledge about the system being simulated; *input uncertainty*, related to uncertainties in the input data; and *structural uncertainty*, associated with the choice of model and assumptions (McKay, Morrison, and Upton, 1999). All these sources can introduce errors in simulation results.

Among these, input uncertainty stands out as a significant contributor to simulation errors, as even slight changes in input can lead to substantial variations in output. Various factors contribute to input uncertainty, with measurement errors being one of the most common. Measurement errors can occur when determining the position, velocity, or other properties of particles within a system. Additionally, imprecise knowledge of the laws of physics can contribute to input uncertainty. For instance, in simulating planetary motion, the gravitational force between planets is typically calculated using Newton's law of gravity, which is an approximation with unknown precision.

While literature contains approaches that specifically address input uncertainty, many of these methods focus on discretely sampling input parameters (Kurniawati et al., 2011; Santolaria and GinéS, 2013). However, these approaches often necessitate multiple simulation instances to draw conclusions, presenting a notable limitation.

3

SIMD OPTIMIZED BOUNDING VOLUME HIERARCHIES

Following the trend of acceleration by parallelization it is obvious to apply this idea also to CD. Unfortunately, the parallelization of the simultaneous traversal for CD is not obvious. Actually, due to their recursive nature, BVHs are not very well suited for massively parallel acceleration on the GPU. Moreover, especially for online planning, robotic agents are often not equipped with powerful GPU.

However, the simultaneous traversal required in BVH-based CD can still benefit from the SIMD instruction sets of modern Central Processing Units (CPUs).

3.1 SIMD RECAP

Originally, SIMD instruction sets had been introduced to support integer computation for intensive multimedia applications, but later they have been extended to support floating point computation which extends the usefulness also for scientific computations. The idea is that a single instruction operates on different input data values (e.g. 4, 8 or 16 floating point values) simultaneously (See Figure 3.1).

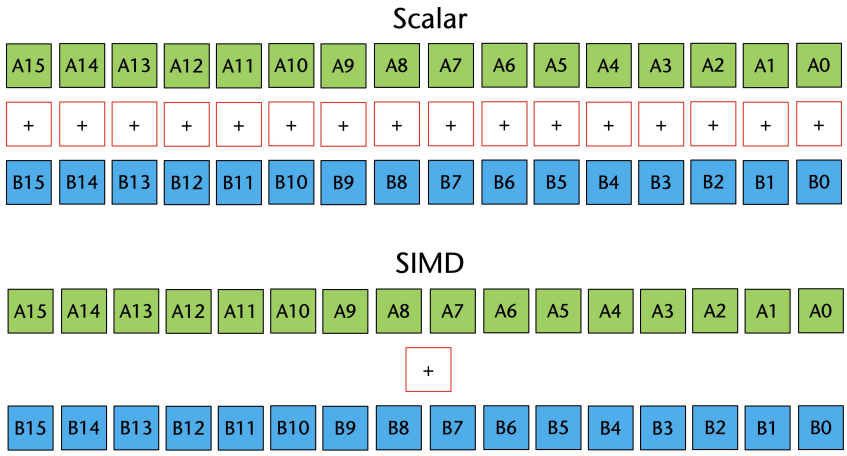


Figure 3.1: Vectorized operations on SIMD architectures support multiple input data values simultaneously in a single operation.

Several slightly different SIMD instruction sets are available for various CPUs; e.g. NEON for Arm based CPUs and Streaming SIMD Extensions (SSE)/Advanced Vector Extensions (AVX) for both Intel and AMD CPUs (see Table 3.1 for a list of available SIMD instruction sets and the supported data types). The latest 512-bit Advanced Vector Extensions (AVX-512) instruction set supports computation of 16

single precision-float in parallel (see Figure 3.2). In this thesis, the implementation is done with AVX-512, however the idea can be easily implemented on other SIMD instruction sets such as SSE, AVX, and NEON.

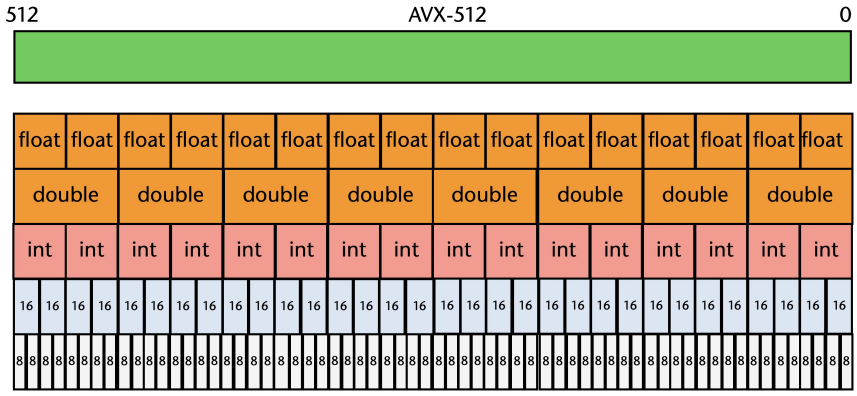


Figure 3.2: The latest AVX-512 registers, with a width of 512 bits, support the operation of 16 floating-point values at one time.

Name	Width	Types	supported CPUs
NEON	128 bits	4x single 2x double*	Armv7-A/R and above *only available for Armv8-A
SSE	128 bits	4x single	Intel Pentium III and above AMD Athlon XP and above
SSE2 SSE3 SSE4	128 bits	4x single 2x double	Intel Pentium 4 and above AMD Athlon XP and above
AVX AVX2	256 bits	8x single 4x double	Intel Sandy Bridge and above AMD Bulldozer and above
AVX-512	512 bits	16x single 8x double	Intel Skylake-X and above

Table 3.1: Floating-point support for various SIMD architectures

3.2 IMPLEMENTATION STRATEGIES

To take advantage from parallel operations offered by SIMD, we can:

1. simply switch on a compiler option and hope that the compiler will do the optimization,
2. optimize the *traversal* function manually, depending on the chosen BVH,
3. or adapt the complete BVH structure which additionally requires a redesign of the BVH *construction*.

In this thesis we have implemented and tested all of these three possible implementations. There is only one suitable function in algorithm 1 to optimize the traversal without changing the tree structure: the test for intersection of a pair of BVs. Hence, the benefit of SIMD optimization relies heavily on the type of the BV. For two spheres, we simply have to compute the distance of two points and compare it to the sum of the spheres' radii. This is not very well suited for the for SIMD parallelization because of the length of current AVX-512 registers that are able to store 16 floating point values. As a consequence, the intersection test for two spheres can be hardly optimized for SIMD. Similarly, the intersection test for AABB requires four comparisons. Modern AVX-512 registers compare 16 float value in a single instruction and this number will increase with upcoming CPU generations. Hence, these BVs could benefit only from the third method, an optimized BVH, but hardly from a simple optimization of the traversal.

Consequently, we decided to use a BV that naturally supports all three methods: the k-DOP. Basically, k-DOP are an extension of AABB to arbitrary orientations (Zachmann, 1998). They offer a natural trade-off between tightness of the BV and computation time for the intersection test. They show comparable performance to other kinds of bounding volumes (Trenkel, Weller, and Zachmann, 2007). By choosing the number of orientations k according to the SIMD instruction set, it is straightforward to adapt this BV-type to further SIMD developments.

However, this simple SIMD-parallelization still tests only two BVs in one instruction (see Figure 3.4a). Hence, it can be applied to almost all existing k-DOP-based BVHs that typically rely on a binary tree. However, we can also parallelize it in a way that one BV of the first BVH is tested simultaneously against *all* children of the other BVH (see Figure 3.4a).

In order to take full advantage of SIMD in this case we additionally have to change the branching factor of the tree. This is non trivial because traditional BVH construction methods, like the Surface Area Heuristic (SAH), median-, or mid point-split, that assign the primitives into the sub-trees are not suitable for higher branching factors. Consequently, we have developed new BVH construction methods, this includes simple heuristics but also a new method that is based on Batch Neural Gas (BNG) clustering. The advantage of such n-ary trees is not only the SIMD accelerated traversal. Additionally, we get less children than with binary trees and the children are also smaller.

To summarize, the main idea of SIMD optimized data structure is to construct BVHs with higher branching factor that can be later used during run-time in a SIMD optimized traversal algorithm. Hence, the core is the *construction* that is typically done in a *pre-processing* step.

3.3 BVH CONSTRUCTION BASED ON BATCH NEURAL GAS CLUSTERING ALGORITHMS

Usually, a BVH is constructed in a *pre-processing* step that can be computationally more or less expensive. Basically, there exist two major strategies to build BVHs: *bottom-up* and *top-down*. The *bottom-up* approach starts with elementary BVs of leaf nodes and merges them recursively together until the root BV is reached. A very simple merging heuristic is to visit all nearest neighbours and minimize the size of the combined parent nodes in the same level (Roussopoulos and Leifker, 1985). Less greedy strategies combine BVs by using tilings (Leutenegger, Edgington, and Lopez, 1997).

However, the most popular method is the *top-down* approach. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. The main problem is to choose a good splitting criterion. A classical splitting criterion is to simply pick the longest axis and split it in the middle of this axis. Another simple heuristic is to split along the median of the elementary bounding boxes along the longest axis. However, it is easy to construct worst case scenarios for these simple heuristics. SAH tries to avoid these worst cases by optimizing the surface area and the number of geometric primitives over all possible split plane candidates (Goldsmith and Salmon, 1987). Originally developed for ray tracing, it is today also used for collision detection. The computational costs can be reduced to $O(n \log n)$ (Wald, 2007; Wald and Havran, 2006) and there exists parallel algorithms for the fast construction on the GPU (Lauterbach et al., 2009). Many other splitting criteria were compared in (Zachmann, 2000).

We decided to use a *top-down* approach for the hierarchy construction. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. Moreover, we use a *wrapped hierarchy* according to the notion of Agarwal et al. (2001), where inner nodes are tight BVs for all their leaves, but they do not necessarily bound their direct children. Compared to layered hierarchies, the big advantage is that the inner BVs are tighter. The main challenge is to choose a good splitting criterion especially, because traditional splitting criteria like SAH do not work for n-ary trees. We propose splitting criteria based on clustering algorithms, namely BNG.

The Neural Gas (NG) clustering algorithm was initially proposed with the task of minimizing the distortion error of vector quantization coding (Martinetz, Berkovich, and Schulten, 1993). NG does not require any prior knowledge of the set of data points and quickly converges to near optimal distortion errors.

The originally proposed idea iterates through all data points until a stop criterion is met. First, for every presented data vector v, a

neighborhood-ranking is determined, which means, that the prototypes w_i are sorted by their distance to v into $(w_{i_0}, w_{i_1}, ..., wi_{N-1})$.

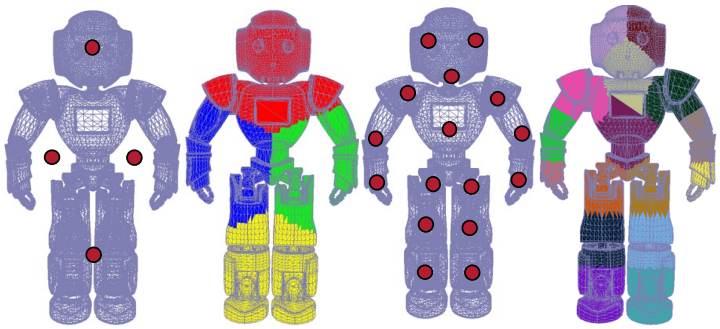
Therefore, for every $k \in \{0, ..., N-1\}$, there are k prototypes w_j with $||v - w_j|| < ||v - w_{i_k}||$.

Let us denote $k = k_i(v, w)$ for every prototype, which depends on both v and the whole set of prototypes w. Then, each prototype w_i is adjusted by:

$$\Delta w_i = \epsilon \cdot h_\lambda(k_i(v, w)) \cdot (v - w_i) \quad i \in 1, .., N \tag{3.1}$$

$\epsilon \in [0, 1]$ represents the overall extent of the modification. The heuristic $h_\lambda(ki(v, w))$ is used to describe the amount of change for each prototype, where λ denotes a constant. Martinetz, Berkovich, and Schulten (1993) proposes $h_\lambda(ki(v, w)) = e^{k_i(v,w)/\lambda}$. If the constant λ is set to zero, Equation 3.1 becomes equivalent to the k-means adaptation rule. For every $\lambda \neq 0$, not only the closest, but every other prototype is updated, too. (Martinetz, Berkovich, and Schulten, 1993, p. 559)

The proposed adaptation (Weller and Zachmann, 2011) involves introducing an adaptive version of the algorithm, which halts the iteration process when the movement of the prototypes becomes smaller than a certain threshold represented by ϵ. In this context, the value of ϵ is approximately set to 10^5 times the size of the bounding box. This modification significantly accelerates the hierarchy's construction without substantially compromising the resultant quality (Weller and Zachmann, 2011, p. 7).



(a) BNG 4 (b) BNG 16

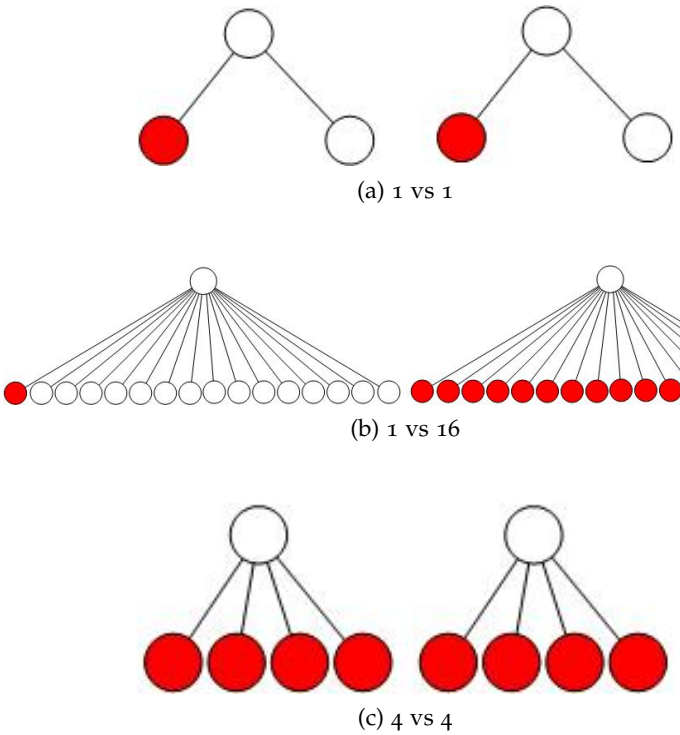
Figure 3.3: 1st Level BVH visualization based on BNG splitting criteria with branching factors of (a) 4 and (b) 16.

Clustering algorithms, especially BNG, have shown to be very efficient for BVH constructions of 4-ary trees (Weller et al., 2014). A nice property of BNG is that it exhibits very robust behavior with respect to the initial cluster center position in contrast to other clustering algorithms like k-means. However, in the original work, the authors used spheres as basic primitives instead of more usual polygonal representations. We simply used the centers of the polygons instead of the

spheres' centers reported in the original work in our polygonal implementation. We did not use magnification control, which additionally considers the size of the spheres to produce better clustering results. However, this can be easily added in the future to our polygon-based BNG.

Figure 3.3 shows the first hierarchy level for BNG splitting with different branching factors.

3.4 SIMD BASED SIMULTANEOUS BVH TRAVERSAL



(a) 1 vs 1

(b) 1 vs 16

(c) 4 vs 4

Figure 3.4: Different strategy to compute intersections of the child nodes in the simultaneous traversal algorithm: (a) classic collision query tests only one pair of nodes. With SIMD, assuming 16 registers, we can (b) test one node of the left BVH against 16 nodes of the right BVH simultaneously in the case of a branching factor of 16 or (c), in case of a branching factor of 4, test all nodes from same level at one time.

The key part to optimize the traversal in algorithm 1 is the test for intersection of the child bounding volumes. For binary trees, the four possible combinations of child pairs are usually traversed sequentially. SIMD enables us to accelerate this intersection test in several ways:

- We can use a SIMD instructions to replace a single test of a pair of BVs (see Figure 3.4a). This would leave the for-loop untouched and just replace the intersection method.

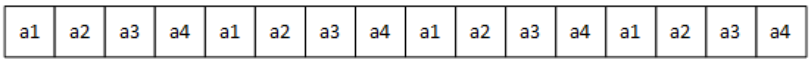
- We can also remove the first part of the for-loop and test one BV of the first BVH simultaneously against *all* children of the other BVH (see Figure 3.4b). For AVX-512 this results in a 1 vs 16 check.
- Finally, we can remove all for-loops and test all nodes from the same level at one time (see Figure 3.4c). With AVX-512 this results in a 4 vs 4 test.

An implementation of the first idea is straight forward, it requires a simple replacement of the comparison inside the intersection function.

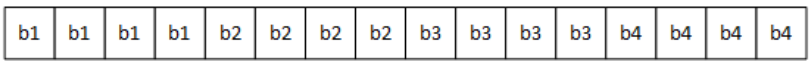
Algorithm 2: _m512 intersect(DOP a, DOP b1,...,b16)

Input : DOP a, DOP $b_1, \ldots, b_{16}$
Output: Intersection result stored in 512-bit vector *endResult*
endResult ← Initialize a 512-bit vector with all elements set to zero
for $i \leftarrow 0$ **to** $k/2-1$ **do**
  oriAL ← Initialize a 512-bit vector with all elements set to $a[i]$
  oriBL ← pack $b_1[i], \ldots, b_{16}[i]$ into a 512-bit vector
  resL ← compare elements in *oriAL* and *oriBL* for less-than condition
  oriAH ← Initialize a 512-bit vector with all elements set to $a[k/2+i]$
  oriBH ← Initialize a 512-bit vector and set element of $b_1[k/2+i], \ldots, b_{16}[k/2+i]$ into it
  resH ← compare elements in *oriAH* and *oriBH* for greater-than condition
  tempRes ← bitwise OR operation between *resL* and *resH*
  endResult ← bitwise OR operation between *endResult* and *tempRes*
  if *all elements of endResult are set to 1* **then**
    Break from the loop
return *endResult*

Removing both loops for the 4 vs 4 test, i.e. testing all n child BVs of object A against all n child BVs of object B for a pair of nodes requires just a slightly different ordering of the Dop values which results in the AVX-512 implementation that is shown in algorithm 8 (See Appendix A 1 for *Intel Intrinsics* code).



(a)



(b)

Figure 3.5: Permutation of the values for the for DOPs $a1, ..., a4$ of an object A and the four DOPs $b1, ..., b4$ of an object B to produce a single 512 Bit AVX register for comparing all 4×4 possible combinations in algorithm 8.

Algorithm 3: _m512 intersect(DOP a1,..,a4, DOP b1,..,b4)

Input : DOP $a_1, \ldots, a_4$, DOP $b_1, \ldots, b_4$
Output: Intersection result stored in 512-bit vector *endResult*
endResult ← Initialize a 512-bit vector with all elements set to zero
for $i \leftarrow 0$ **to** $k/2$ **do**
  oriAL ← pack $a_1[k/2+i], a_2[k/2+i], a_3[k/2+i], a_4[k/2+i]$ into a 512-bit vector according to Figure 3.5a
  oriBL ← pack $b_1[i], b_2[i], b_3[i], b_4[i]$ into a 512-bit vector according to Figure 3.5b
  resL ← compare elements in *oriAL* and *oriBL* for less-than condition
  oriAH ← pack $a_1[i], a_2[i], a_3[i], a_4[i]$ into a 512-bit vector according to Figure 3.5a
  oriBH ← pack $b_1[k/2+i], b_2[k/2+i], b_3[k/2+i], b_4[k/2+i]$ into a 512-bit vector according to Figure 3.5b
  resH ← compare elements in *oriAH* and *oriBH* for greater-than condition
  tempRes ← bitwise OR operation on *resL* and *resH*
  endResult ← bitwise OR operation on *endResult* and *tempRes*
  if *all elements of endResult are set to 1* **then**
    Break from the loop
return *endResult*

3.5 RESULTS

We have implemented our algorithms using C++ and *Intel Intrinsics functions* using *Visual Studio 2017*. We focused our implementation on the most recent AVX-512 instruction sets.

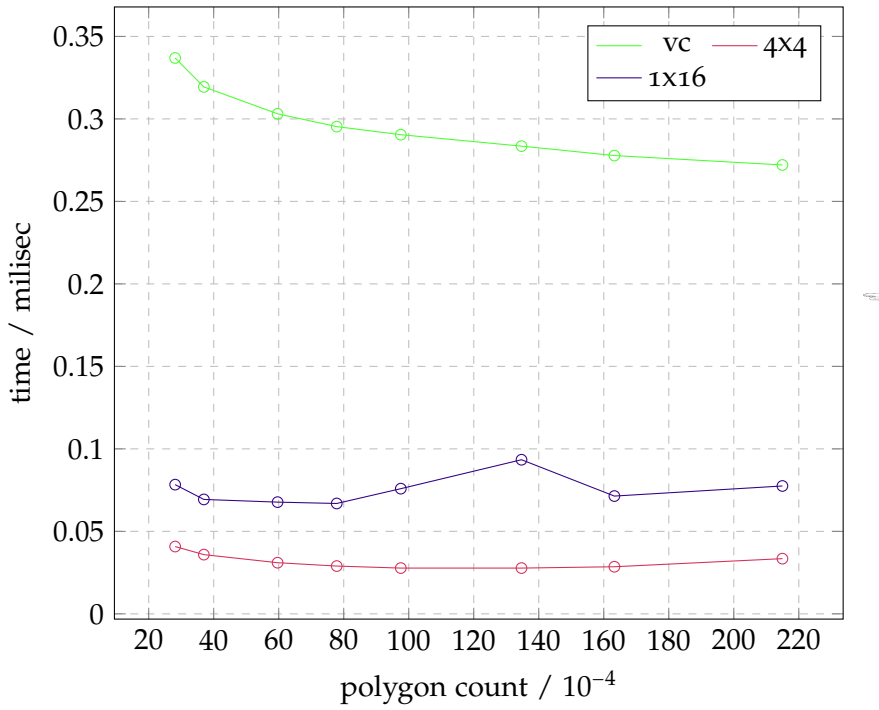
All tests were performed on a system with an Intel I7 7800X CPU, 64GB of main memory and a NVIDIA Geforce GTX 980 GPU with 4GB of memory.

The benefit of the clustering increases with and increasing number of branches in the tree. In term of BVH construction time, the BNG clustering-based SIMDop for both degree of 4 and 16 can be constructed almost as fast BVH constructed using V-COLLIDE (VC) and binary DOP tree (see Figure 3.7).

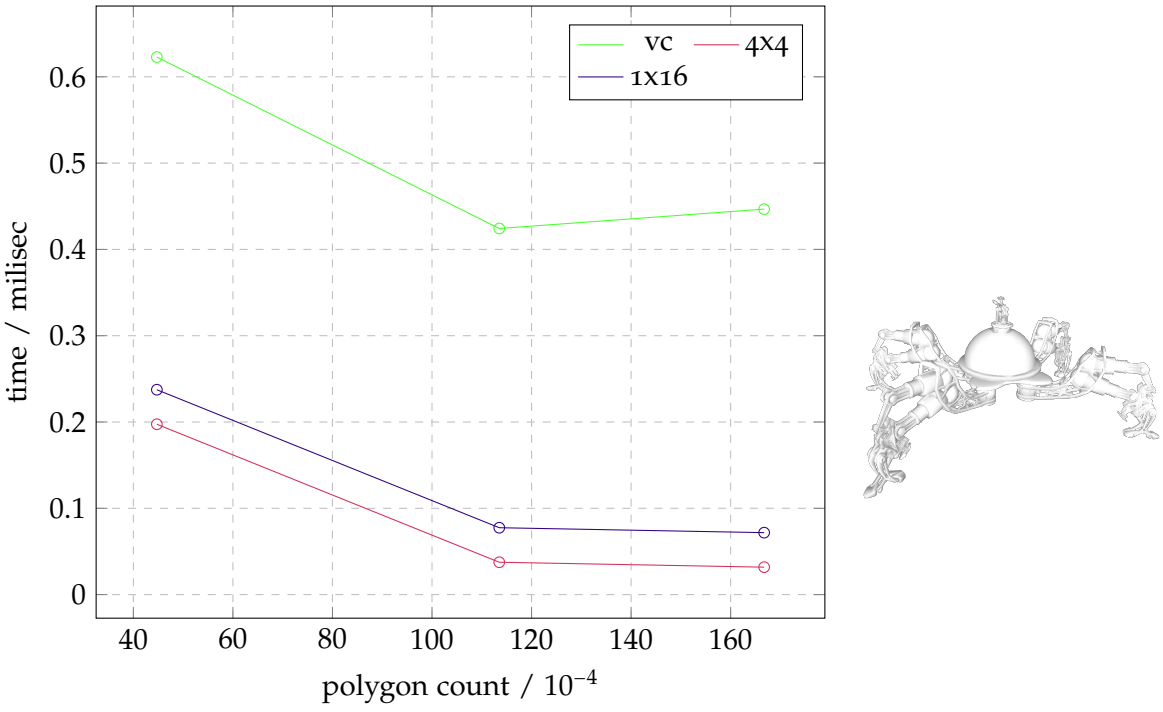
We also compared our SIMDop to the VC library that is often used for sample-based path planning tasks. An experimental comparative analysis has shown that VC outperforms other CD libraries like Proximity Query Package (PQP) (Reggiani, Mazzoli, and Caselli, 2002). Figure 3.6 shows that our 4 vs 4 test is able to outperform VC by a factor of up to 8 for the female robot and up to 13 for quadripod robot object.

3.6 EXTENSION TO CONTINUOUS COLLISION DETECTION

Standard CD techniques that check for collisions at a single point in time, also known as discrete CD, can lead to inaccurate results and

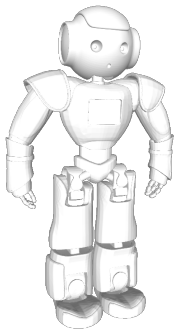


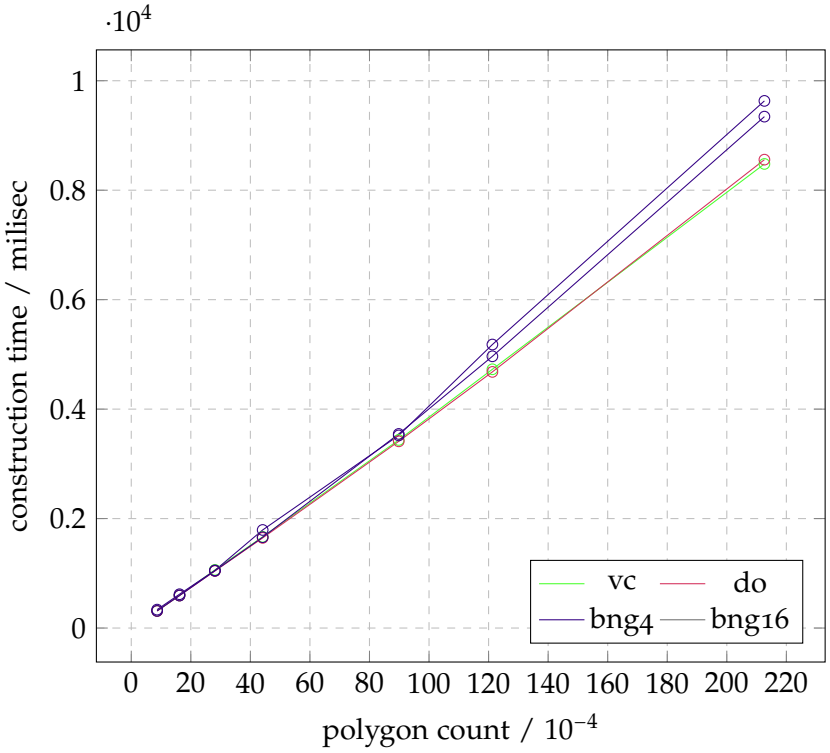
(a)



(b)

Figure 3.6: A comparison of our proposed algorithms with V-COLLIDE library for (a) female robot, and (b) quadripod robot. The results show that our 4 vs 4 test performs best and faster than V-COLLIDE by up to eight times for (a) female robot and by a factor of up to 13 for (b) quadripod robot.





(a)

Figure 3.7: A comparison of BVH construction time of BNG clustering algorithm compared with V-COLLIDE and binary DOP tree for object nao.

unexpected behavior, known as the tunnel problem. The problem arises because standard CD algorithms operate on a per-frame basis, checking for collisions between objects at a single point in time. If two objects are moving quickly and pass through each other between frames, the CD algorithm may not detect the collision, resulting in the objects passing through each other and potentially causing visual glitches or other issues.

For example, consider a game where a player is controlling a fast-moving character that collides with walls and other objects in the environment. If the character moves too quickly, it may pass through a wall without being detected by the CD algorithm, resulting in the player being able to move through solid objects (See Figure 3.8), and potentially breaking the game's mechanics.

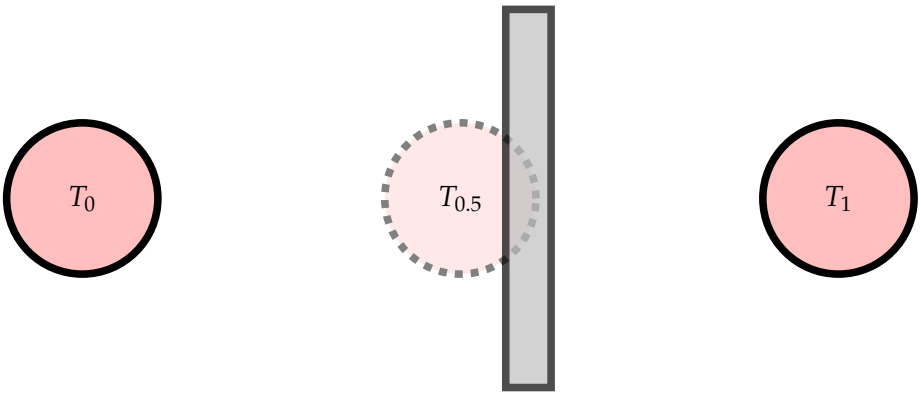


Figure 3.8: The ball's trajectory spans from time T_0 to time T_1, failing to intersect with the grey rectangle, which should have been collided with at time $T_{0.5}$. This occurrence exemplifies a common linear penetration phenomenon resulting from an excessive linear velocity.

One solution to the tunnel problem is to use Continuous Collision Detection (CCD) algorithms, which check for collisions over a range of time rather than at a single point in time. CCD algorithms can detect collisions even if the objects are moving quickly and can prevent the tunnel problem from occurring.

Another approach is to reduce the speed of the objects in the simulation, so that they do not move fast enough to pass through each other between frames. However, this can result in less realistic and engaging simulations, especially in games where fast movement is an essential part of the gameplay.

The concept of CCD, as first applied by Eckstein and Schömer (1999), addresses these issues in exchange for an increase in running time. In CCD, instead of only comparing objects at their respective start and end position, their entire path is considered. Therefore, it is impossible to miss collisions and the exact moment of collision is always determined perfectly.

Held, Klosowski, and Mitchell (1996) introduced a pseudo-CCD approach, employing static CD with smaller time steps to diminish the likelihood of overlooking actual collisions. Despite this improvement, the possibility of missed collisions still exists. To address this issue, Mirtich (2000) proposed the technique of conservative advancement. This method involves iteratively advancing objects by a certain timestep to ensure that no object penetrates another. Additionally, Zhang, Lee, and Kim (2006) presented a method for iteratively computing new maximum limits for advancement based on the object's minimum distance.

A reliable approach to CCD involves enclosing the given BV at the beginning and end of the movement step within a swept volume and then testing for collisions within this volume. This methodology has been explored extensively across various types of bounding volumes. For instance, it has been applied to AABB (Eckstein and Schömer, 1999), velocity-aligned Discrete Oriented Polytope (DOP) with swept volumes based on underlying spheres (Coming and Staadt, 2008), OBB (Redon, Kheddar, and Coquillart, 2002), sphere swept convex hulls (Taeubig and Frese, 2012), and even ellipsoids (Choi et al., 2009).

CCD has also been presented for articulated (Zhang et al., 2007) and deformable (Tang et al., 2008) models. Additionally, a technique for interactive CCD for topology changing models has been proposed by He et al. (2015). Redon, Lin, and Manocha (2004) presented an CCD approach for articulated models combining BVHs and swept volumes.

Kim et al. (2009a) utilized both the CPU and GPU in a hybrid parallel CCD approach. To support non-linear movements of rigid bodies during CD, Buss (2005) presented a robust approach for CCD using relative screw motion.

Merkt, Ivan, and Vijayakumar (2019) proposed CCD into collision avoidance for robotic agents.

3.6.1 *Inner Sphere Tree*

Weller and Zachmann (2011) presented the novel data structure of Inner Sphere Tree (IST), which was used for discrete CD. The theoretical background for CCD using IST was given as well.

For the linear movement case, a moving sphere creates a capped cylinder. The test between two capped cylinders can be defined as follows:

Let the center move from P_t to P_{t+1} for the first sphere and from Q_t to Q_{t+1} for the second sphere. Let r_p denote the radius of the first sphere and r_q denote the radius of the second sphere. Our task is to determine whether the distance between the line segments (P_t, P_{t+1}) and (Q_t, Q_{t+1}) is less than the sum of the radii r_p and r_q. Instead of dealing with intersecting two capped cylinders, our focus lies on tracking the movement of two spheres along their respective line segments.

Given that both spheres move with the same relative normalized distance toward their respective destinations, we can make further simplifications. Thus, the closest distance is given by:

$$r_{toi} = \pm\sqrt{\frac{r^2 - \Delta P^2}{\Delta m^2} + \left(\frac{\Delta P \cdot \Delta m}{\Delta m^2}\right)^2 - \frac{\Delta P \cdot \Delta m}{\Delta m^2}} \tag{3.2}$$

with $r = r_p + r_q$, $\Delta P = P_t - Q_t$, $m_p = P_{t+1} - P_t$, $m_q = Q_{t+1} - Q_t$, and $\Delta m = m_p - m_q$.

To compare two spheres for collision, both are usually transformed into their respective coordinate systems, which would result in two matrix multiplications. However, this can be reduced to only one matrix multiplication. If the second object is simply not transformed at all, the first object needs to be transformed by the inverse of the second object's model matrix to stay relatively transformed. This reduces the number of matrix multiplications per sphere to only one.

Let S_p and S_q be the two spheres, $M_{t,p}$ and $M_{t,q}$ their starting transformation matrices, and $M_{t+1,p}$ and $M_{t+1,q}$ their destination transformation matrices. Then the matrices $M_{t,pq}$ and $M_{t+1,pq}$, which can be used to transform S_p into the same coordinate system as S_q, and $M_{t,qp}$ and $M_{t+1,qp}$ analogously, can be calculated as follows:

$$M_{t,pq} = M_{t,p} \cdot M_{t,q}^{-1} \tag{3.3}$$

$$M_{t+1,pq} = M_{t+1,p} \cdot M_{t+1,q}^{-1} \tag{3.4}$$

$$M_{t,qp} = M_{t,pq}^{-1} \tag{3.5}$$

$$M_{t+1,qp} = M_{t+1,pq}^{-1} \tag{3.6}$$

As we never actually need to transform the second sphere, its start and end positions are the same. Therefore, m_q always equates to zero, which is why we can replace Δm with m_p, resulting in the following equation:

$$r_{toi} = \pm\sqrt{\frac{r^2 - \Delta P^2}{m_p^2} + \left(\frac{\Delta P \cdot \Delta m}{m_p^2}\right)^2 - \frac{\Delta P \cdot m_p}{m_p^2}} \tag{3.7}$$

The result must be clamped into the range $[0, 1]$, ensuring that the closest distance is actually determined within the range of the two line segments. Also, if both spheres move along parallel paths, Δm is zero. This case must be handled separately in the implementation.

3.6.2 *BVH Construction*

Slightly different from Section 3.3, the original IST take the volume of the spheres into account during BVH construction. The idea is based on magnification control (Hammer, Hasenfuss, and Villmann, 2007), which allows to add weighting factors to increase the space in some areas and thus consider the actual size of the given spheres. Implementing the weighting factors $u(v)$, equation Equation 3.1 becomes:

$$\Delta w_i = \epsilon \cdot h_\lambda(k_i(v,w)) \cdot u(v) \cdot (v - w_i) \quad i \in 1,..,N \tag{3.8}$$

As we desire to consider the volume of the spheres, the weighting factor is simply set to $u(v) = \frac{4}{3}\pi r^3$ (Weller and Zachmann, 2011).

3.6.3 *Results*

In our test setup, we placed two dragons close to each other and moved until they swap positions. There is no overlap in the beginning or end of the simulation, but they fully overlap in the middle. (see Figure 3.9)

Applying SIMD based simultaneous traversal from Section 3.4 improve the collision timing by up to 2x for branching factor 4 and up to 3x for branching factor of 16 when both objects are heavily collised given a test setup describen in Figure 3.9.

Both implementations are suitable for faster-than-realtime scenario.

3.7 CONCLUSION AND FUTURE WORK

We have presented two versions for a SIMD optimized bounding volume hierarchy for simultaneous BVH traversal. The main idea is to use higher *n-ary* trees instead of classical binary trees. We also presented BNG-based clustering algorithm for the top-down construction of such tree data structures with higher branching factor. Even if we tested only up to 16-ary trees, the clustering-based construction is already prepared to support higher branching factors following future SIMD developments.

Our results show that, depending on the object, our SIMD-based BVHs outperform traditional BVHs by more than an order of magnitude for static CD.

In case of CCD, applying our proposed SIMD-based traversal improves the collision timing by up to 3x during our test.

Our approach also opens up several directions for future work. For instance, we would like to include magnification control to the BNG construction algorithm for static CD. Moreover, other clustering algorithms than BNG could be considered. In this work, we relied on DOP as BV because of a fair comparison with the manual optimized

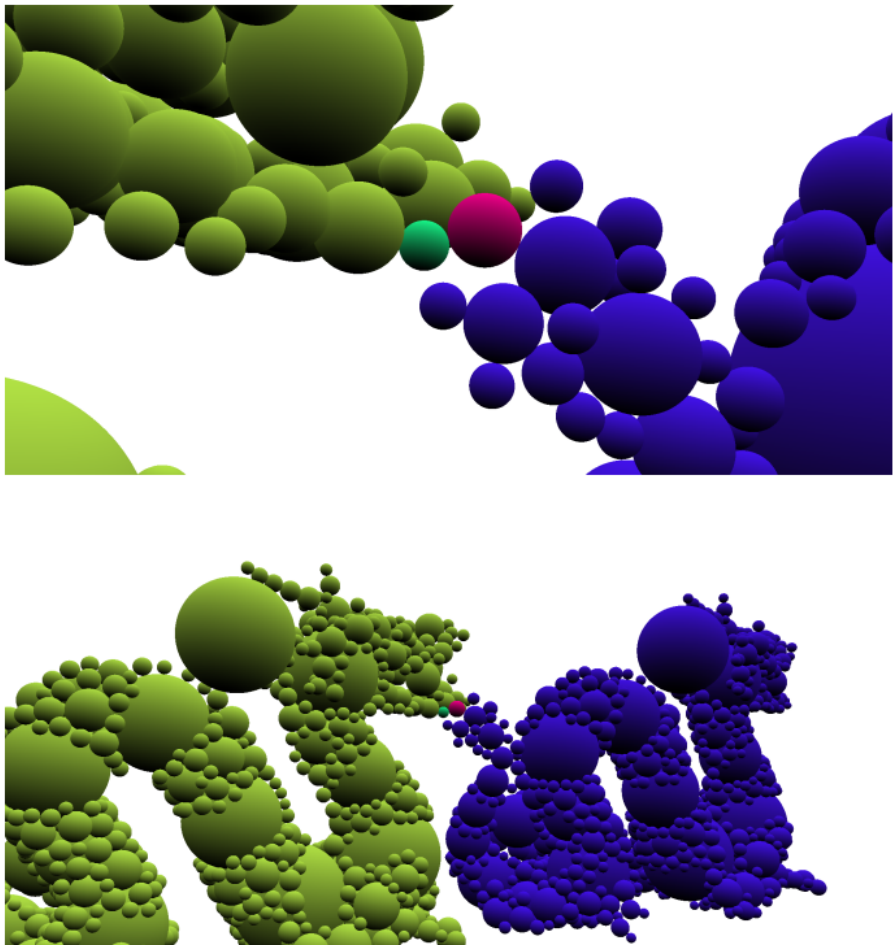


Figure 3.9: Two Stanford dragons are placed close to each other and moved until they swap positions. There is no overlap in their hierarchy at the beginning or end of the simulation, but they fully overlap in the middle, enabling evaluation of various traversal depths. Each time step is repeated 127 times to compute the median performance, reducing variance significantly. (Groß, 2022)

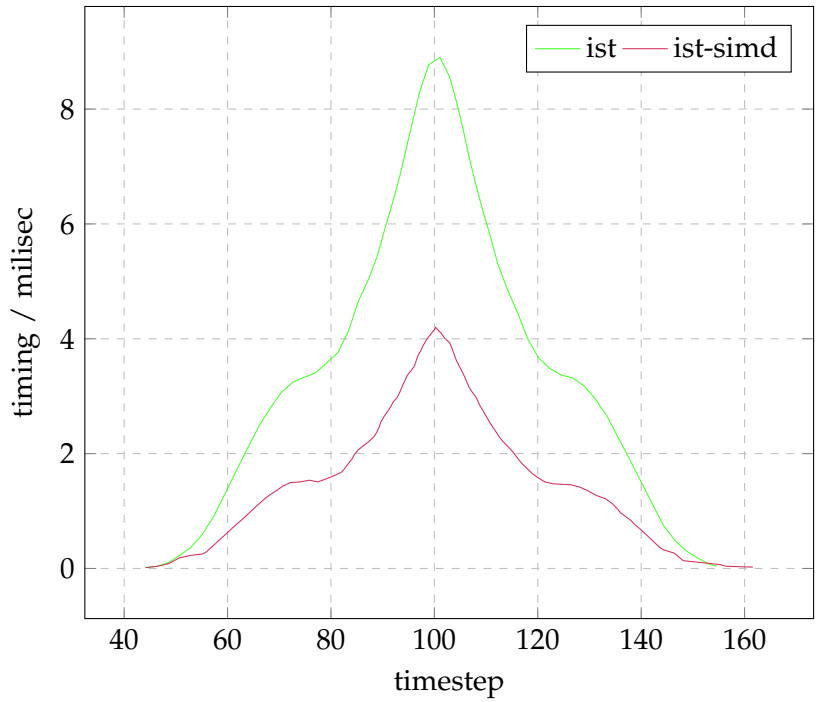


Figure 3.10: The median collision check timing for the test scenario described in Figure 3.9 is calculated for the BVH with a branching factor of 4 and SIMD-based simultaneous traversal, as explained in Section 3.4.

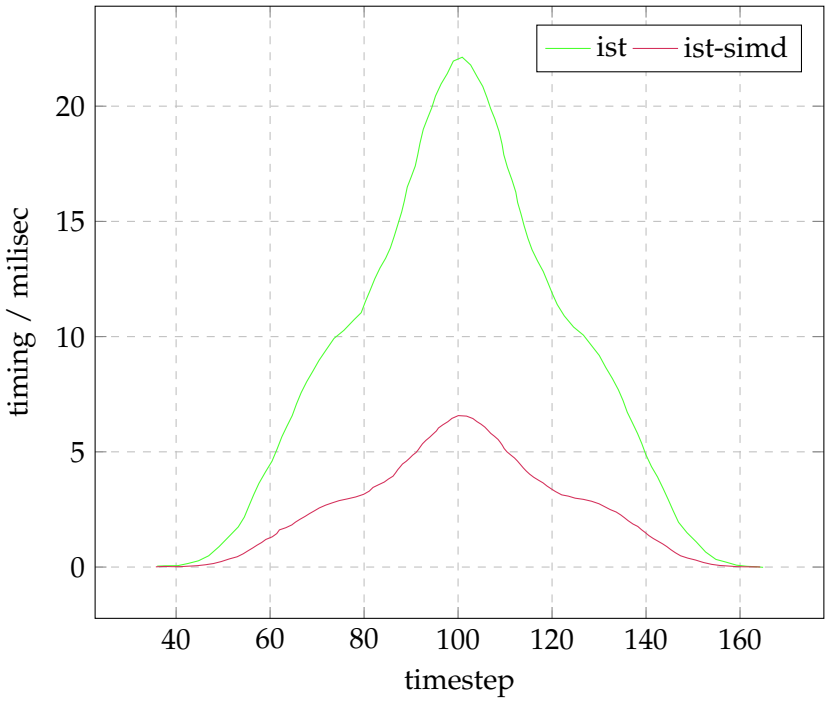


Figure 3.11: The median collision check timing for the test scenario described in Figure 3.9 is calculated for the BVH with a branching factor of 16 and SIMD-based simultaneous traversal, as explained in Section 3.4.

traversal scheme. However, we would like to investigate also other BV types that do not have the problem of the later escape of the for-loop. Also the influence of the number of orientations for the DOP requires further investigations. Finally, probably other applications using BVH like ray tracing or occlusion computations could benefit from our proposed BVH too.

4

MEMORY EFFICIENT BOUNDING VOLUME HIERARCHIES

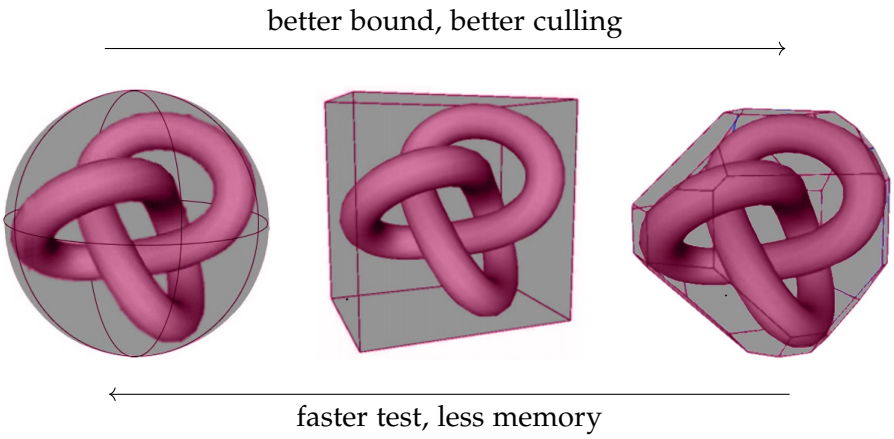


Figure 4.1: Complex BVs such as (c) k-DOP can often provide a better fit for models, resulting in improved performance compared to less tight BVs like (a) Sphere or (b) AABB. However, it's important to note that the trade-off for this improved fit is a larger memory footprint.

Chapter 3 presents an order of magnitude acceleration compared to existing algorithms such as PQP or VC. However, despite the performance, BVHs-based collision detection also suffers from a disadvantage: *memory footprint*. While a more complex BV like k-DOP can generally provide a better fit for models and thus yield better performance compared to less tight BV like Sphere or AABB, it comes at the cost of a larger memory footprint.

BVs	Descriptor	Memory Footprints
Sphere	center & radius	4 floats
AABB	min & max	6 float
k-DOP	number of k	k float

Table 4.1: Commonly used bounding volumes for BVH and their memory requirements.

This can be observed in Table 4.1, which showcases the varying memory requirements for some common used BV types.

For applications like ray tracing, to avoid out-of-core techniques during large-scale simulations that do not fit into the system's main memory, some approaches are employed. These approaches include reducing BVH precision (Vaidyanathan, Akenine-Möller, and Salvi, 2016),

using half-precision floating-point (Koskela et al., 2015), compressing leaf nodes with multiple polygons (Benthin et al., 2018), representing meshes using alternative representations (Lauterbach, Yoon, and Manocha, 2007), or utilizing implicit BVH representations (Bauszat, Eisemann, and Magnor, 2010).

In order to handle large-scale simulations without resorting to out-of-core techniques, Kim et al. (2009b) propose a method that involves quantifying BV descriptors and partitioning the BVH into smaller sub-trees for collision detection purposes. To ensure random access to BVHs, they compress and store sub-tree information in hash tables.

On the other hand, for general CD scenarios, such as those found in mental simulations, Koskela et al. (2015) employs half-precision floating-point representation (16-bit float). Tan, Weller, and Zachmann (2019) utilize SIMD Instruction Sets, specifically the *_mm512_cvtps_ph* instruction, to convert 32-bit floating-point values into half-precision floating-point representations for the BV descriptors. This conversion reduces the memory requirements from 4 bytes to just 2 bytes per value. However, since SIMD does not support direct operations on half floats, the BV descriptors need to be converted back to full precision using the *_mm512_cvtph_ps* instruction during BVH traversal. This approach instantaneously reduces the memory footprint by half, without sacrificing performance.

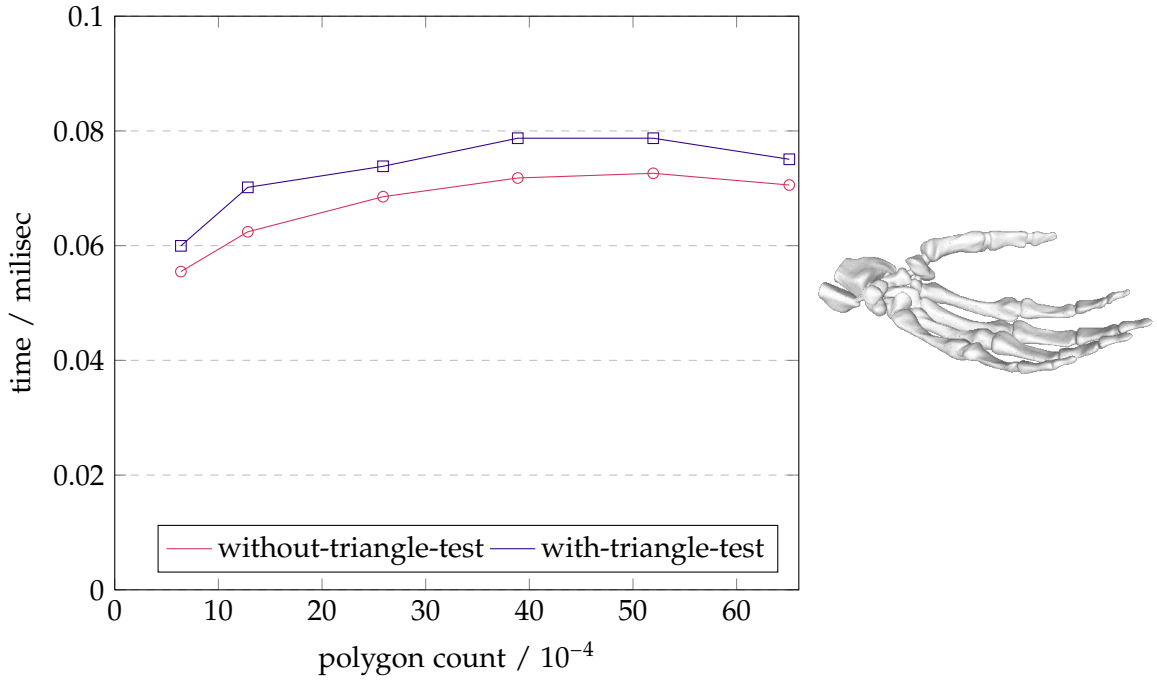
While 16-bit floating-point representation offers a significant reduction in memory usage, storing BV descriptors beyond that limit is currently not feasible due to the absence of SIMD instruction sets designed for such purpose. Moreover, as the compression level increases, the reconstructed BVs may suffer from substantial loss of precision. Another possibility to reduce memory footprint is by eliminating leaf nodes, as they can account for up to 50% of the memory footprint in the case of complete binary BVHs. This proportion becomes even higher when dealing with higher branching factors. By removing leaf nodes, significant memory savings can be achieved.

Surprisingly, the impact of reducing memory footprint is often overlooked in the literature concerning CD. Most authors tend to prioritize the evaluation of their CD algorithms based on their running time, with little attention given to memory considerations. To the best of our knowledge, there is currently no research exploring quantification beyond 16-bit for CD purposes.

This is primarily due to the fact that most models typically involve a relatively small number of polygons, or the environments consist of only a few objects. However, the situation can be quite different for robotic mental simulations, where multiple instances need to run simultaneously. While it is possible to offload computationally expensive and resource-intensive simulations to cloud computing, as discussed in Bozcuoğlu and Beetz (2017), the utilization of sophisticated and memory-efficient CD algorithms can greatly enhance the

number of simultaneous simulations that can be performed, leading to improved performance (Viitanen et al., 2017). This efficiency becomes even more critical for online robotic agents operating in remote areas, where access to cloud computing resources may be limited or unavailable. By optimizing memory usage in CD algorithms, the capability for running multiple simulations concurrently can be significantly increased, enabling more efficient and autonomous robotic systems in resource-constrained environments.

In this chapter, our focus is on optimizing the memory footprint of BVH used by existing algorithms, namely *Doptree* and *Boxtree* algorithms.



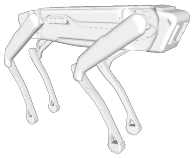
(a)

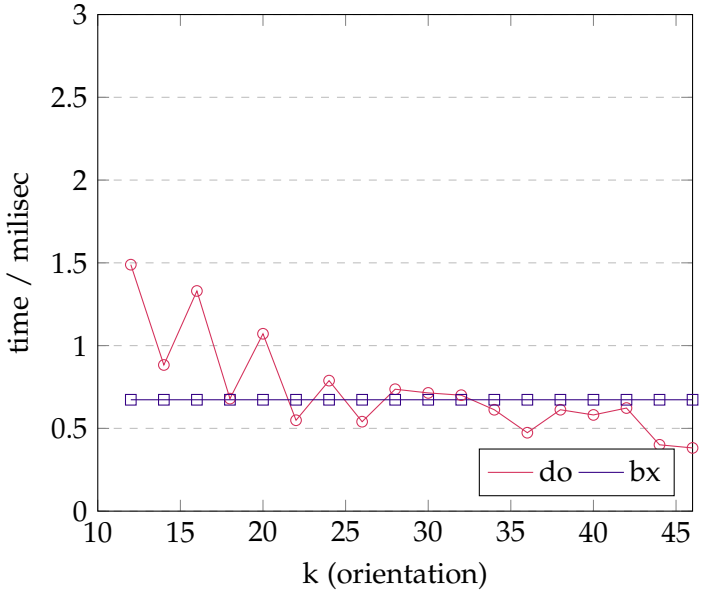
Figure 4.2: Average collision query time for the object Hand with and without triangle test for Doptree with $k = 46$

To provide a quick comparison of their performance for CD, Figure 4.3 is presented. It demonstrates that, in scenarios involving higher orientation, *Doptree* outperforms *Boxtree* in terms of performance. However, it comes with the trade-off of a larger memory footprint. On the other hand, *Boxtree*, known for its space subdivision approach, is widely recognized for its memory efficiency.

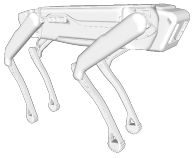
4.1 MEMORY EFFICIENT DOPTREE

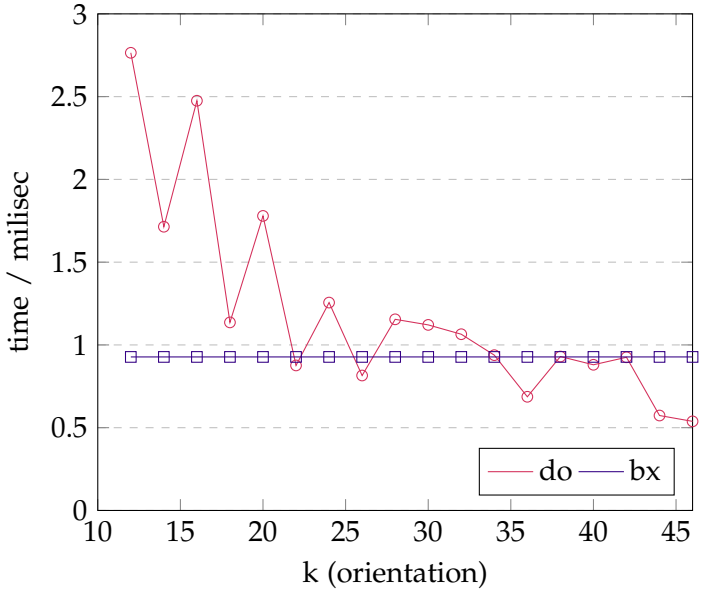
In general, it is expected that certain parts of the BV are redundant, as some parts of the child BVs may share the same information as the parent BV (See Figure 4.4).





(a)





(b)

Figure 4.3: A comparison between Doptree and Boxtree algorithms for various Doptree orientations with (a) single collision option, and (b) all collision option for object spot

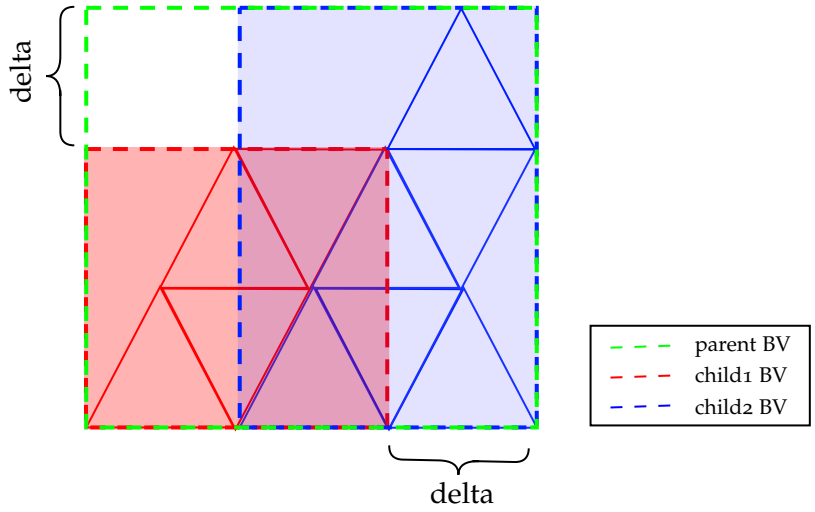


Figure 4.4: The calculation of the delta for bounding volumes (BVs) is based on the parent-child relationship

In our benchmark, we analyzed the percentage of redundant information for different objects, which varied from 38.74% for the object ds9 to as high as 87.00% for the object eagle. Table 4.3 presents the percentage of redundant orientations obtained using BNG on k-DOP with 24 orientations for BVH construction.

Object	Redudant BV Ori
ds9	38.74 %
armadillo	42.47 %
eagle	87.00 %
ferrari	75.66 %
grid	57.67 %
happy-buddha	44.20 %
robot-dog	42.92 %

Table 4.2: Percentage of Redundant BV Descriptor for various objects using BNG for BVH construction.

The percentage of redundant information remains consistent regardless of the number of orientations, as illustrated in Table 4.3. This table specifically showcases the results for object ds9, which consists of 282,499 polygons. The construction process yielded 139,942 inner nodes and 282,499 leaf nodes.

4.1.1 *Optimization*

In the case of Doptree, we chose to compress by quantification. Beside delta values for each orientation, we calculated additional data re-

Nr Ori	BV Memory (MBytes)	Redudant BV Ori
6	9.90	41.84%
12	19.80	38.78%
24	39.60	38.74%
36	59.40	38.85%
46	75.90	38.82%

Table 4.3: Percentage of Redundant BVs Descriptor for object ds9 using BNG during BVH construction.

	Min	Max	d[0]	d[1]	...	d[k-1]	d[k]	∑
DOPTree	-	-	32	32	...	32	32	(k * 32) bits
BVHLite	16	16	x	x	...	x	x	(k * x) + 32 bits

Table 4.4: Doptree BV Descriptor

quired for the later reconstruction of the BVs, specifically the minimum and maximum delta values. These values are presented in Table 4.4.

To facilitate the quantification process, we rounded the deltas either up or down based on the original BV descriptor. The rounding operation ensures that the reconstructed BVs are slightly larger to avoid missing collisions.

During the traversal process, before performing the BV overlap test (as expressed in Equation 4.4), we need to reconstruct the BVs by reversing the quantification process. This reconstruction step is essential for accurately determining BV overlaps and conducting collision tests.

$$interval = \frac{|\Delta_{max} - \Delta_{min}|}{bits} \tag{4.1}$$

if $d[k]_{ori} > 0$:

$$d[k] = \left\lfloor \frac{\Delta_k - \Delta_{min}}{interval} \right\rfloor \tag{4.2}$$

else:

$$d[k] = \left\lceil \frac{\Delta_k - \Delta_{min}}{interval} \right\rceil \tag{4.3}$$

$$d[k]_{ori} = d[k]_{parent} - (d[k] * \frac{|\Delta_{max} - \Delta_{min}|}{bits} + d_{min}) \tag{4.4}$$

To accommodate the changes, we need to adjust the BVH traversal algorithms (See algorithm 4).

Algorithm 4: BVHLiteTraversal(BV *a_parent*, BV *b_parent*, BV *a*, BV *b*)

```
if a and b are both leaves then
    checkPrimitives(a, b)
else if a is leaf then
    forall children b_i of b do
        reconstructBV(b_i_parent, b_i)
        if a and b_i intersect then
            BVHLiteTraversal(a_parent, a, b_i_parent, b_i)
else if b is leaf then
    forall children a_i of a do
        reconstructBV(a_i_parent, a_i)
        if a_i and b intersect then
            BVHLiteTraversal(a_i_parent, b_parent, a_i, b)
else
    forall children a_i of a and b_i of b do
        reconstructBV(a_i_parent, a_i)
        reconstructBV(b_i_parent, b_i)
        if a_i and b_i intersect then
            BVHLiteTraversal(a_i_parent, b_i_parent, a_i, b_i)
```

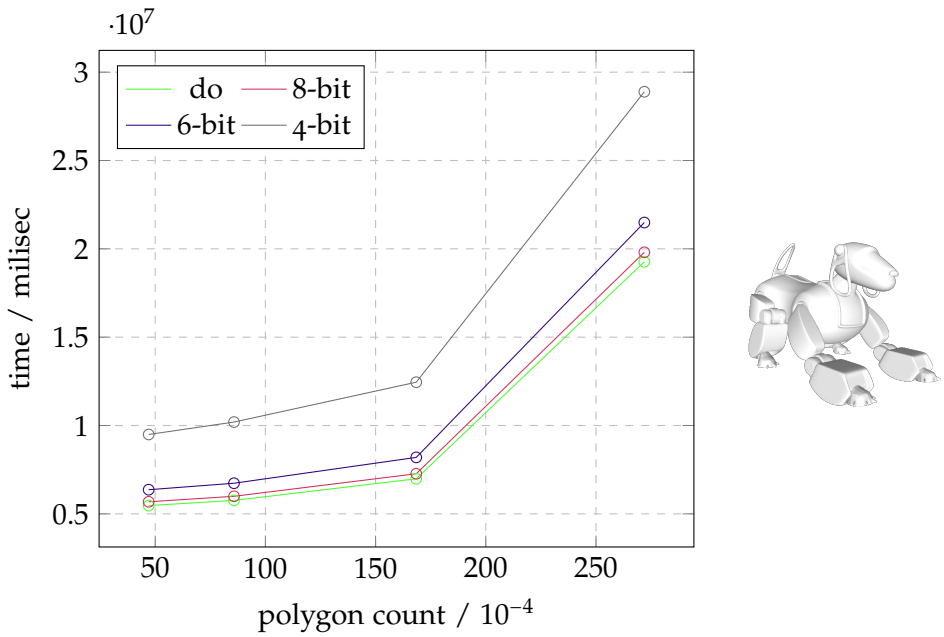


Figure 4.5: Median collision check timing for object Robot Dog for Doptree algorithms and various compression levels.

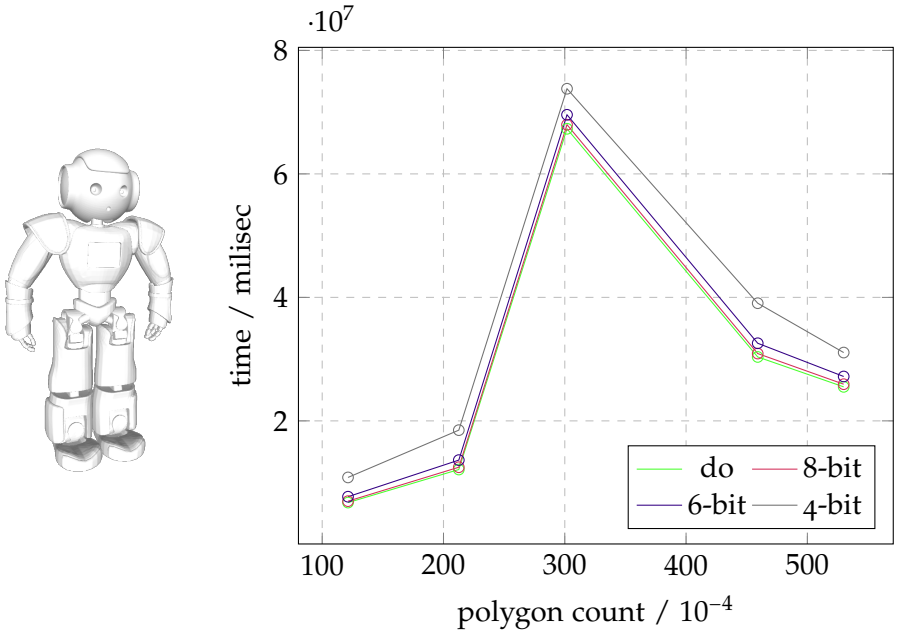


Figure 4.6: Median collision check timing for object Nao for Doptree algorithms and various compression levels.

4.1.2 *Results*

As expected, by pushing the BV descriptor to use as few bits as possible, the timing will suffer since it needs to reconstruct BV before each overlap test and the reconstructed BVs are slightly bigger (See Figure 4.6 & Figure 4.5). In comparison with the original implementation, using 6 bits seems reasonable with little slow down. Beyond that, the collision check timing will suffer greatly. The original implementation of Doptree with a branching factor of 4 requires 1600 bits to represent internal node (32 bits * 46 orientations) + (4 * pointer to child), whereas our approach using 6-bit quantification and implicit pointer only requires 340 bits (6 bits * 46 orientations) + (1 * pointer to child).

4.2 MEMORY-EFFICIENT BOXTREE

The Boxtree algorithm (Zachmann, 1995) relies on using boxes as bounding volumes. It starts with a single box that wraps around the entire object. Through a repeating process, this box is split into two new boxes. This division happens until each box contains only one polygon. At this stage, the boxes hold crucial information about the enclosed polygons, essential for performing polygon intersection operations. The structure is a binary tree, where regular boxes are nodes, and boxes enclosing polygons are leaves.

The original implementation is relative straightforward. Each box is a separate heap object, serving as either a node or a leaf within

the tree structure. Nodes store crucial information about subsequent boxes' placement and orientation, represented by two floating-point values and an integer denoting the cutting plane.

Nodes also maintain two pointers to the next boxes, where a Null-pointer indicates an empty sub-box.

In contrast, leaf nodes in the Boxtree implementation solely store information related to the corresponding polygon, including the polygon's index and the four possible indexes of its constituent points. As the actual 3D point coordinates are stored in the geometry, a pointer to this array is also stored alongside the indexes for coordinate retrieval. Although leaf nodes have an integer for the cutplane, it's not utilized in the present context.

To differentiate between nodes and leaves, both using a union to accommodate both sub-types, the upper bits of the cutplane are used as a marker to indicate the type.

The theoretical representation of each number as 4 bytes suggests a total size of 32 bytes. However, practical considerations introduce additional factors, including an unused virtual destructor, padding requirements, and the behavior of the malloc function, which may allocate more bytes than strictly necessary. While this over-allocation is specific to the implementation and may not be consistent across all compilers or Standard Libraries, it's a noticeable consequence of this particular implementation style observed on multiple machines. Detecting this over-allocation can be done by observing unexpectedly high memory usage and confirmed using the malloc usable size() function, providing information about the actual allocated bytes per allocation.

Considering all factors, such as the unused virtual destructor, padding requirements, and possible over-allocation by the malloc function, the size of a Boxtree box nearly doubles. Consequently, the total size of a Boxtree box is 56 bytes. Therefore, the resulting representation of a Boxtree box is as follows:

4.2.1 *Optimization*

Even though Boxtree inherently exhibits memory efficiency, there remains an opportunity for further enhancement.

4.2.1.1 *Moving Constant Numbers*

Each leaf in the Boxtree holds a 64-bit pointer pointing to its geometry points. Although this pointer could be redundant, it is necessary because every main Boxtree object requires all the details to compare against another Boxtree. To optimize the redundant pointers, we add a header to the overall Boxtree. This header becomes the first point of reference when comparing two Boxtrees. It allows us to store constant

Boxtree	
virtual destructor 8 cutting plane 4 padding 4	
Union	
Node	Leaf
left-node-ptr 8 right-node-prt 8 cutp 4 cutr 4 padding 8	polygon-index 4 padding 4 4x point-index 16 points ptr 8
malloc overhead 8	

Table 4.5: Original Boxtree structure on a x64 system (size in Bytes) (Schwochow, 2021)

information once and pass it to the checking method during collision checks.

4.2.1.2 *Indexing*

In Boxtree, nodes and leaves are allocated as heap objects. Each node requires two 64-bit pointers to the next heap object, as well as an additional 64-bit for every allocation. To optimize this, we can store nodes and leaves in an array. Instead of using 64-bit pointers, only 32-bit indices are needed. The resulting array will be stored in and read from the header.

4.2.1.3 *Merge Leaves*

Additionally, it is also possible to reduce the number of references, especially to leaves, by merging leaves with their parent nodes. This results in four different types of nodes, depending on the amount and placement of the merged leaves.

The ratio of original nodes to optimized nodes is almost the same across various objects and resolutions (See Table 4.6).

4.2.1.4 *Minimize Leaves*

The leaves consist of a pointer to an array of 3D coordinates, or points, of the geometry, which is assigned during BVH construction and used during collision checks. However, there is also the index of the polygon in each leaf, which refers to the polygon in the geometry. With this, the entire polygon can be read at runtime from the geometry, as long as the pointer to the polygon array in the geometry is stored. The only difference between the polygons in the geometry and those in

the leaves is that during construction, the last of the indexes in the leaves would be marked when the polygon is a triangle. However, this information can also be obtained directly from the polygon pointer, reducing the size of a leaf to just one integer representing the polygon index. As nodes now either have an index to refer to another node or refer to a polygon with an index, these two can be represented by a single union, resulting in the same size for all node types. For this to work, the type of the index has to be stored, which will be done in the cutplane integer that is not fully used. Having the same size allows the use of only one array to hold all nodes, without any space wasted for padding.

Object	Original Nodes	Optimized Nodes	%
ATST-4252	11623	7419	63,83
ATST-152944	417155	266924	63,98
Castle-14871	39630	25011	63,11
Raptor-40000	1172783	772783	65,89

Table 4.6: Nodes count in comparision to original implementation.

4.2.1.5 *Cuts*

The only part of the Boxtree that hasn't been optimized yet is the cut information in every node. This includes the cutplane orientation (x, y, z) and two floats marking where the box is split. Because this info is exactly what's gained when building the Boxtree and is essential for every check, it's trickier to optimize.

To compress the floats, the simplest choice is to use 16-bit floats, which are converted to and from 32-bit floats when stored and read. Converting from 16-bit floats involves just 3 instructions Section A 2 and maintains the same performance level.

4.2.2 *Optimized Structure*

With these optimizations, the resulting Boxtree achieves a size reduction of about 71% compared to the original representation, resulting in a size of 16 bytes per node. This memory-efficient design enables multiple nodes to fit within a single cache line without overlap. In typical systems, a cache line represents 64 consecutive bytes of memory, read and cached as a whole for improved performance Drepper, 2007.

The precise layout of the four resulting Boxtree types is as follows:

These optimized Boxtree types ensure efficient memory usage and alignment, maximizing cache utilization for improved performance during memory access and traversal operations.

	cutplane	1. type	2. type	2x indices	2x 16-bit floats	total
two leaves	4	-	-	8	4	16
left leaf	4	0	-	8	4	16
right leaf	4	-	0	8	4	16
no leaf	4	0	0	8	4	16

Table 4.7: Optimized Boxtree structure on a x64 system (size in Bytes) (Schwochow, 2021)

4.2.3 *Results*

The optimized implementation of the Boxtree introduces changes that are expected to result in minimal performance penalties. The most significant impact is likely the removal of polygons from the leaf nodes. Instead of accessing the polygons from the node itself, they need to be read from the polygon vector in the geometry, resulting in additional memory accesses. Converting a 16-bit float to a 32-bit float typically requires only three instructions using Intel Intrinsics, but these instructions can accumulate over the runtime. On the other hand, moving constant information into a header has almost no performance cost since this information is likely to be cached after accessing the first node.

Storing nodes in an array can potentially improve performance by allowing multiple nodes to be on the same cache line. This increases the likelihood of the next accessed node already being in the cache, resulting in faster access compared to reading from memory. To maximize this effect, the first child of each node is placed immediately after the parent node, and the checking function starts with that child whenever possible.

Considering a node size of 16 bytes and a cache line size of 64 bytes, it is not possible to cache the next three nodes by just reading the first node.

Despite the potential improvements, the memory-optimized version is still approximately 10% slower than the original implementation (see Figure 4.7). As expected, the average speed reduction can be attributed to the float conversion and array access, with each contributing around 3-5% to the total runtime. However, in terms of cache performance, the optimized version exhibits fewer cache misses overall, despite the additional read into the geometry. This indicates that the multiple nodes per cache line are extensively utilized and contribute to maintaining relatively similar overall performance.

Although the optimized version cannot match the performance of the original implementation, this minor performance loss is insignificant compared to the significant memory gain of approximately 82%.

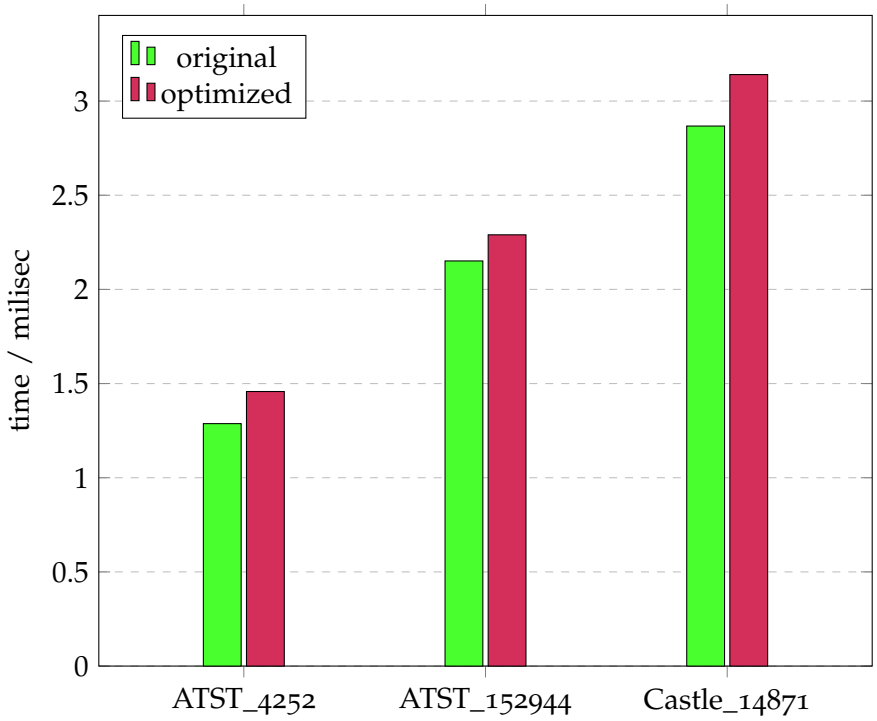


Figure 4.7: Comparison of original speed vs. optimized

4.3 CONCLUSION AND FUTURE WORK

We have introduced approaches aimed at reducing the memory footprint of two existing CD algorithms based on BVHs, namely Doptree and Boxtree. Our implementation atop the Doptree algorithm has slashed the memory usage to a mere 5.31%, while for the Boxtree algorithm, it stands at 10% of the original algorithms' memory consumption. This enhancement empowers robotic agents to significantly increase the number of concurrent mental simulations they can execute.

While the optimized Doptree tends to outpace Boxtree in speed, it does so at the cost of consuming up to four times more memory compared to the optimized Boxtree. To illustrate, consider an object with 400,000 polygons: in Boxtree, inner nodes would require 11 bytes, whereas Doptree, with 6 bits quantization, requires 41 bytes. In our assessment, both algorithms are equally viable for mental simulations.

This also opens up several directions for future work. For instance, we could consider another compression method based on a predictor-corrector schema (Solenthaler and Pajarola, 2009). Another approach might involve approximating entire BVHs using polynomial regression instead of explicitly storing BV nodes. However, this approximation might require additional pre-processing steps to ensure it fits into the BVHs adequately.

Another interesting direction for future work could involve integrating uncertainty into mental simulations. For instance, by returning

the continuous distribution instead of discrete final states from the simulation. This approach will significantly reduce the number of simulations needed for decision-making.

Exploiting temporal coherence (Ponamgi, Manocha, and Lin, 1995) between simulations could also be considered to further accelerate mental simulations.

5 BENCHMARKING AS ONLINE SERVICE

Benchmarking is a critical aspect of algorithm evaluation, providing valuable insights into the performance and capabilities of various methodologies. However, this process can be arduous and complex, accompanied by its fair share of challenges. These challenges include *dataset heterogeneity*, where the availability of diverse and representative datasets can lead to disparate evaluation results, making it difficult to draw meaningful conclusions about algorithm performance. Additionally, *task complexity* demands multiple benchmarking metrics to accurately capture various aspects of algorithm performance, posing a challenging endeavor in identifying suitable and effective metrics. Another challenge lies in the *limited ground truth*, as the absence of a reliable reference standard makes it difficult to gauge the true accuracy of algorithms, leading to less robust evaluations. Moreover, *replication crisis* is also a growing concern in scientific research, and the field of computer graphics is no exception. It refers to the inability to reproduce research findings, leading to doubts about the reliability and validity of published results. Several factors contribute to this crisis, including the lack of detailed instructions, missing data or codes, and compatibility issues with hardware and software.

In a recent study conducted in the domain of computer graphics, the extent of the replication crisis was examined (Bonneel et al., 2020). The study analyzed 374 papers from prestigious conferences like the Association for Computing Machinery (ACM) Special Interest Group on Computer Graphics and Interactive Techniques (SIGGRAPH) 2014, 2016, and 2018, and the results shed light on the challenges faced by researchers in replicating and validating findings. Out of the 374 papers, 151 software packages made available for replication. Of 151 software packages, only 133 came with complete source codes, while the remaining 18 were pre-compiled softwares. A considerable 68 of these source codes required modification to work properly, while 19 encountered technical issues, and 5 were hampered by hardware compatibility problems. These challenges in replicating the results highlight the pressing need for more transparent and standardized practices in computer graphics research.

5.1 OPEN BENCHMARKING FOR REPRODUCIBLE AND COMPARABLE RESULTS

To avoid replication crisis and enhance the credibility of research findings, it is crucial to establish an open benchmark that fosters repro-

ducibility and comparability of results. Such a benchmark would serve as a standard platform for researchers to share their algorithms, data, and codes in a clear and accessible manner. By providing detailed instructions and openly sharing their work, researchers can significantly enhance the chances of successful replication and validation by the broader community.

Key requirements for an open benchmark include:

- **Reproducible and Comparable Results:** The foremost requirement for an open benchmark is to ensure that researchers can reproduce each other's experiments accurately. By following standardized procedures and sharing comprehensive documentation, the benchmark facilitates comparability and aids in validating the robustness of the proposed methodologies.
- **Same Hardware and Software Environment:** To maintain consistency and eliminate hardware or software-related discrepancies, an open benchmark should mandate the use of the same hardware and software environment for all experiments. This ensures that results are not influenced by variations in computing resources and allows for a fair evaluation of different approaches.
- **Easy-to-Use Interface:** an open benchmark must be designed with user-friendliness in mind. An intuitive and user-friendly interface makes it accessible to researchers with diverse backgrounds and expertise. By simplifying the process of data submission and result analysis, the benchmark encourages active participation and collaboration.
- **Sustainability:** sustainability is a key aspect of an open benchmark. To ensure its long-term viability, the platform should be well-maintained and regularly updated to accommodate emerging research trends and new hardware or software advancements. Continuous support and development are vital for the benchmark's relevance and usefulness.

To maximize accessibility and reach, an open benchmark can be established as an Online Service . This approach allows researchers from different locations to access the benchmark, i.e., through web-based interfaces. Benchmark as Online Service (BOS) offers several advantages, including ease of access, comparable data analysis, and scalability. Researchers can submit their experiments, analyze results, and compare their findings with others in a collaborative and efficient manner.

5.2 BENCHMARKING FOR CD & PQ ALGORITHMS

Benchmarking BVH-based CD & PQ algorithms serves as a perfect example of addressing the challenges in benchmarking. Algorithms

are using different BV, from spheres (Hubbard, 1996), AABBs (Bergen, 1998; Zachmann, 1995), k-DOPs (Klosowski, 1998; Zachmann, 1998), OBBs (Gottschalk, Lin, and Manocha, 1996), spherical shells (Krishnan et al., 1998), to swept spheres (Larsen et al., 2000).

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, Weller, and Zachmann, 2019).

The reason for such a large amount of different CD & PQ algorithms 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. Frequent Operating System (OS) updates could also lead to dependency hell (See Figure 5.1). 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 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.

5.2.1 *Web-Based Benchmarking Service*

In this dissertation, we realize the idea of open benchmark by running CD & PQ benchmark as an online service. The idea is to simplify the complex and time-consuming process of benchmarking CD & PQ algorithms, more precisely, of methods for the broad phase CD & PQ of rigid polygonal polygon soups.

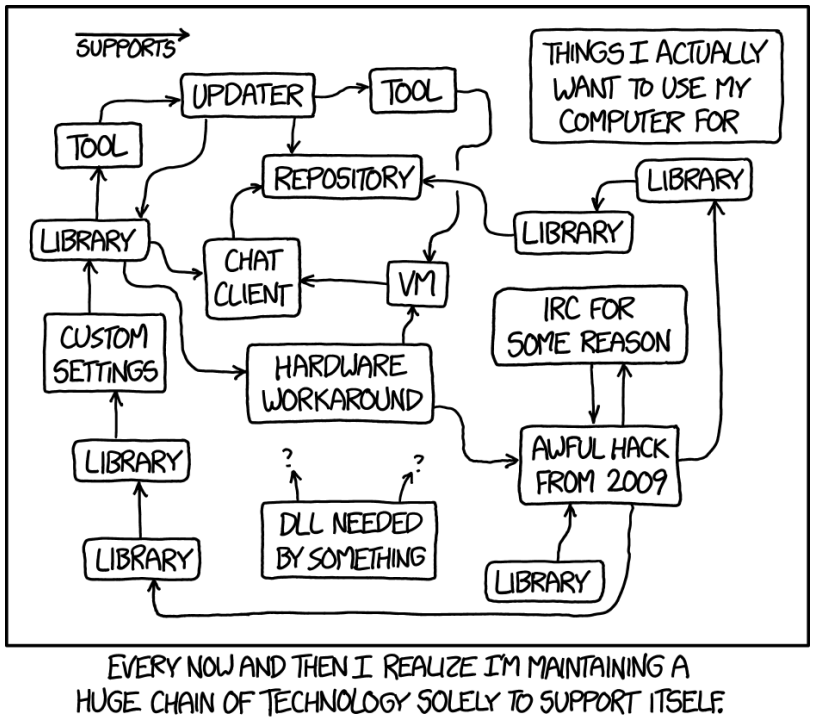


Figure 5.1: OS updates can disrupt software functionality, leading to a cascade of required updates and compatibility issues, known as "dependency hell". (xkcd.com, 2024)

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.

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.

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, Weller, and Zachmann, 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.

5.2.2 *Heatmap Visualization*

The benchmarking suite by Trenkel, Weller, and Zachmann (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 5.2b). 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.

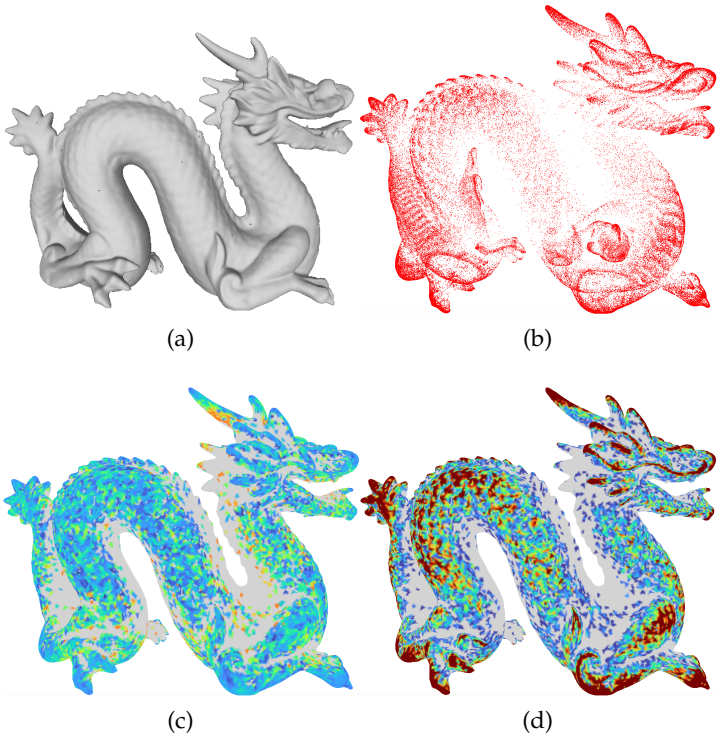


Figure 5.2: Heatmap generation pipeline based on benchmark's result: (a) 3D object, (b) closest points of all configurations, (c) generated heatmap based on algorithm timings, and (d) generated heatmap based on configurations density.

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 5.3a), median (Figure 5.3b), min (Figure 5.3e), and max (Figure 5.3f) timing.
    - to visualize critical regions based on algorithm's timing.
2. Standard Deviation (Figure 5.3c) and Median Absolute Deviation (Figure 5.3d)
    - to visualize outlier regions where algorithm's timing could differs greatly between slightly different configurations.
3. Configuration density (Figure 5.3g)
    - to visualize regions that are extensively or hardly checked by algorithms.

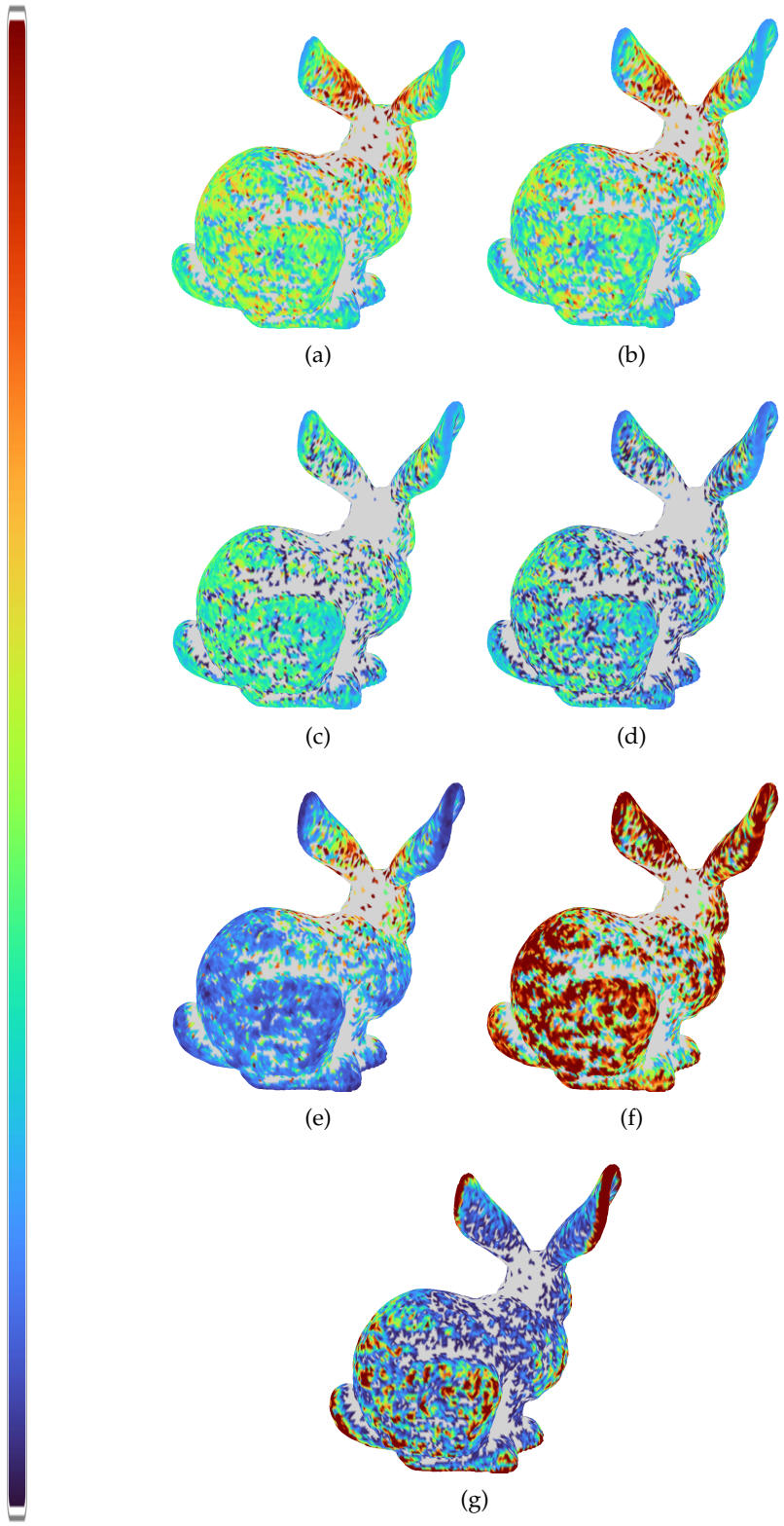


Figure 5.3: An example of heatmap based on configuration's (a) average timings, (b) median timings, (c) standard deviation timings, (d) median absolute deviation timings, and (e) min timings, (f) max timings, and (g) density.

A primary goal of openCollBench for benchmarking 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 5.7 shows an overview of our system; it is based on a front end that provides an easy-to-use Graphical User Interface (GUI) to the user and a dedicated back end server that performs the actual benchmarking.

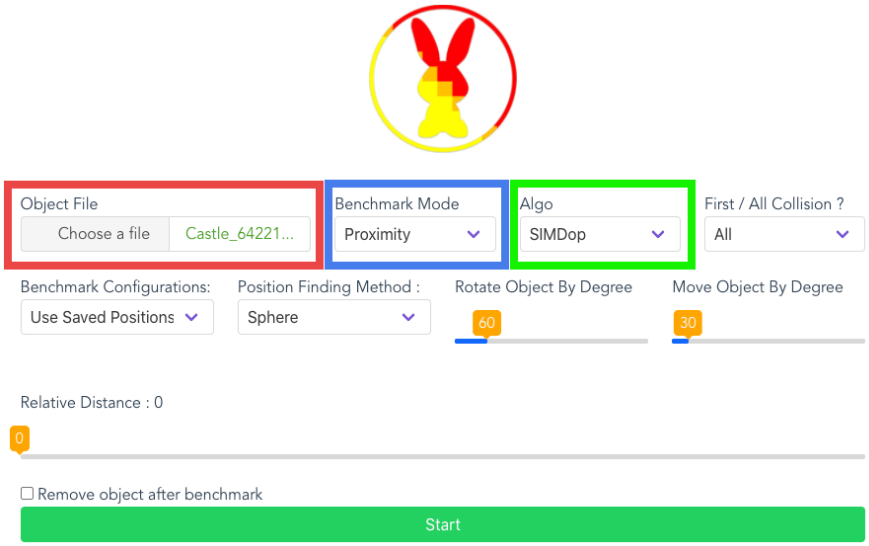


Figure 5.4: Interactive Graphical User interface (GUI) for OpenCollBench, which enable user to upload object (red box) and selecting benchmark parameters interactively. The option panels connected with each other, i.e. changing *Bench Mode* (blue box) to proximity will display algorithms that support promxity query in *Algo* (green box).

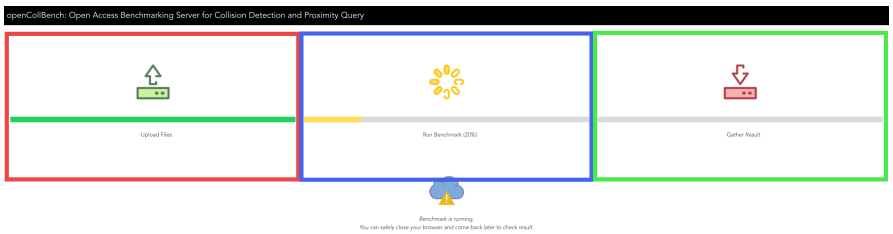


Figure 5.5: Benchmark's progress GUI for OpenCollBench, which consists of three parts, namely, *left* (red box) showing progress of object uploaded by user, *middle* (blue box) showing benchmarking progress including configurations generation, and *right* (green box) showing progress of heatmap generation pipeline.

The front end is designed to focus on simplification and usability of the benchmarking process to enable both expert and non-expert users

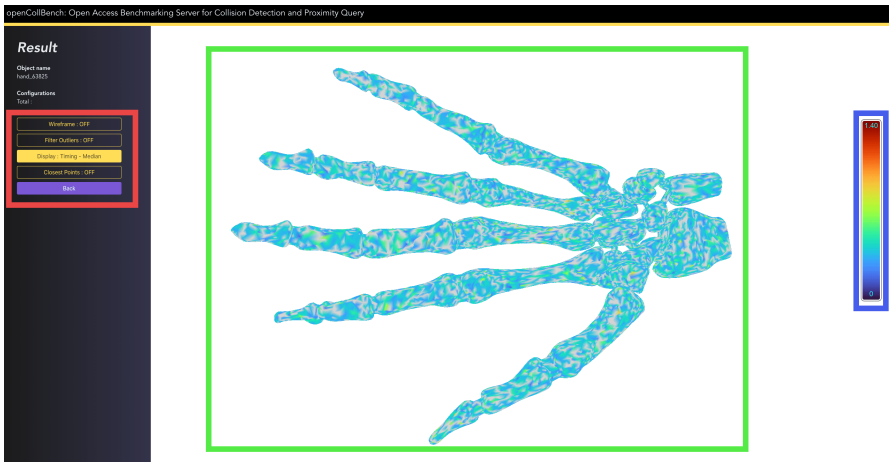


Figure 5.6: Benchmark's results GUI for OpenCollBench, which showing generated heatmap based on benchmarking results. Left panel (red box) enable user to select different mapping value, middle panel (green box) showing generated heatmap, and right panel (blue box) showing mapping color value.

to intuitively benchmark CD algorithms. We have implemented our front end using the vue.js framework. Figure 5.4 shows the website to select appropriate benchmark parameters via sliders 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 5.5) 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 5.6).

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, Weller, and Zachmann (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.

5.2.3 *Safe Execution of User-Uploaded Algorithms*

One challenge of providing benchmarking as an online service is the integration of new algorithms into the existing benchmarking tool due to security concerns of running unknown code, which will always pose a risk, i.e., Remote Code Execution (RCE). Directly analyzing and validating the code is not trivial, not to mention authors might not want to disclose their algorithms in some cases. Hence, it does make sense to run user-uploaded algorithms in case of doubt in an environment where it can not cause any damage. This could be done on another physical computer accessible over the network and that does not have access to critical systems and does not contain sensitive data. However, the fact that the machine is connected to other computers in a network is already a risk. It is also challenging to identify whether this system is compromised.

This is where the implementation of hardware virtualization becomes advantageous. Here, access to the physical machine's hardware is regulated by a so-called *hypervisor*. This can be an OS that runs natively on the hardware (Type 1), e.g., *Microsoft Hyper-V, VMWare ESXi* or software that runs on an operating system and simulates hardware access (Type 2), e.g., *Microsoft Virtual PC, Oracle Virtual Box, VMware Workstation*. A Virtual Machine (VM) can be started via the hypervisor, which operates completely isolated from the underlying systems. In addition, a virtual network can be configured with the hypervisor, to which only the host system and the virtual operating system have access. This means the virtual system has no access to external networks to which the host system is connected. On top of that, many hypervisor implementations offer a so-called *Snapshot* function that

can save the state of the virtual machine at a specific point in time and restore it if necessary. This resets all data changed over the runtime, both on the virtual storage medium and the data in the virtual main memory.

In order to guarantee the security of such a scenario, we shift the execution of user-uploaded algorithms into a virtualization environment. It is sufficient for authors to compile and upload their proposed algorithms as wrapper Dynamic-Link Library (DLL) specified by *openCollBench*.

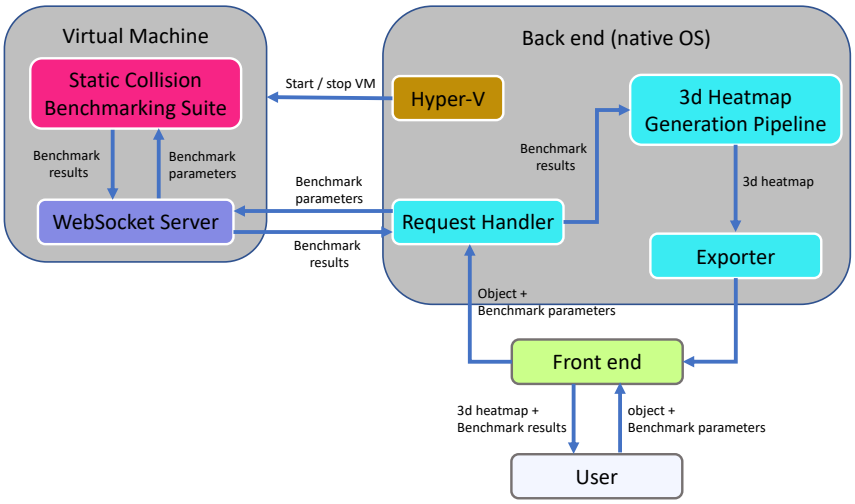


Figure 5.7: System Overview of openCollBench

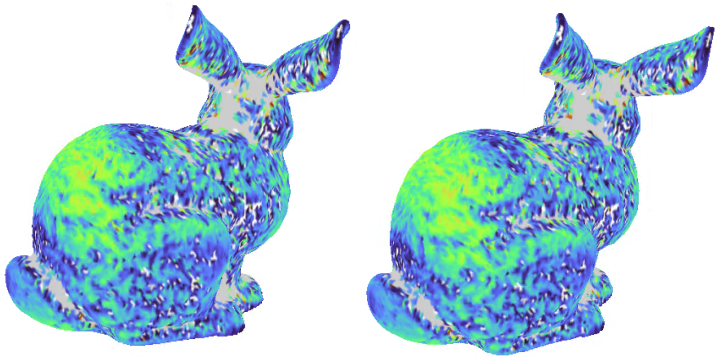


Figure 5.8: Average collision query time heatmap for the object Bunny in native (left) and virtualization environment (right) using boxtree algorithm on Intel CPU (i7 7900x). The heatmaps are very similar.

5.3 CONCLUSION AND FUTURE WORK

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

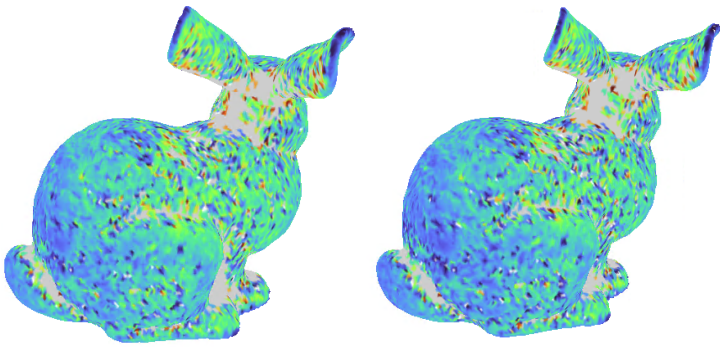


Figure 5.9: Average collision query time heatmap for the object Bunny in native (left) and virtualization environment (right) using vcollide algorithm on Intel CPU (i7 7900x). The heatmaps are very similar.

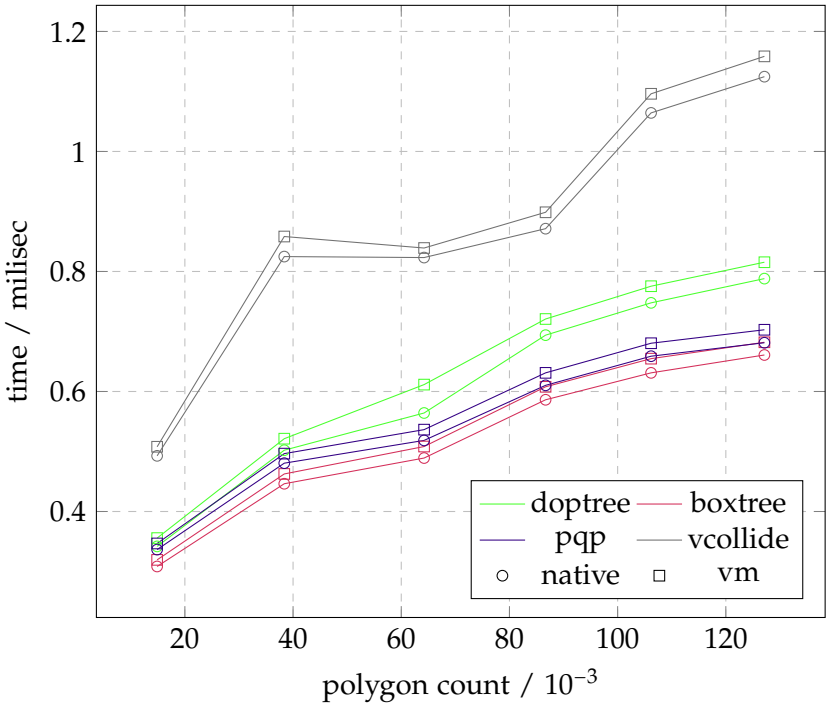


Figure 5.10: Average collision query time for the object Castle in native (○) and virtualization (□) environment for various CD algorithms using AMD CPU (Ryzen 9 3900X). The delta are very similar across different algorithms, object shapes and complexity.

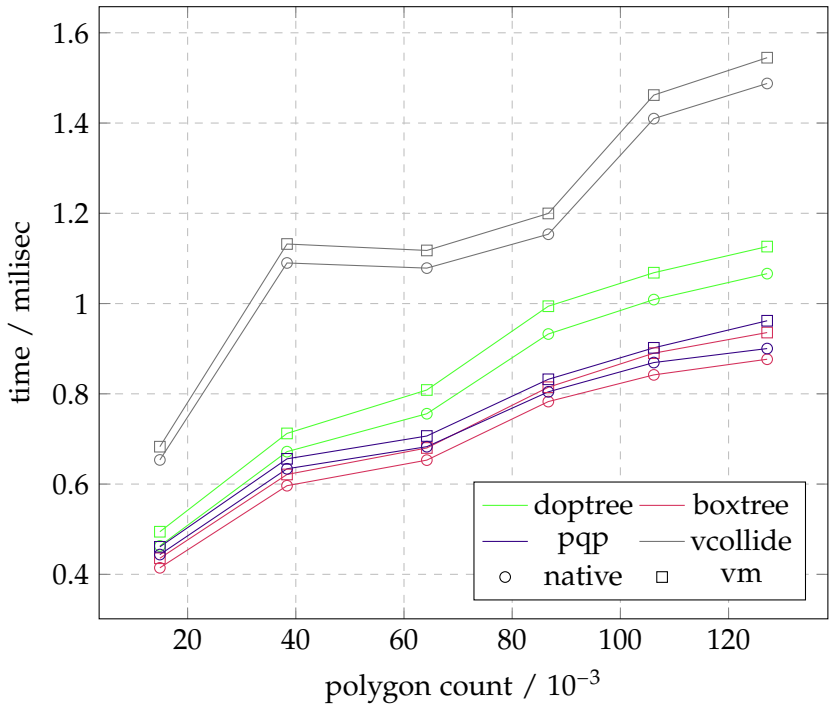


Figure 5.11: Average collision query time for the object Castle in native (○) and virtualization (□) environment for various CD algorithms using Intel CPU (i7 7900x). The delta are very similar across different algorithms, object shapes and complexity.

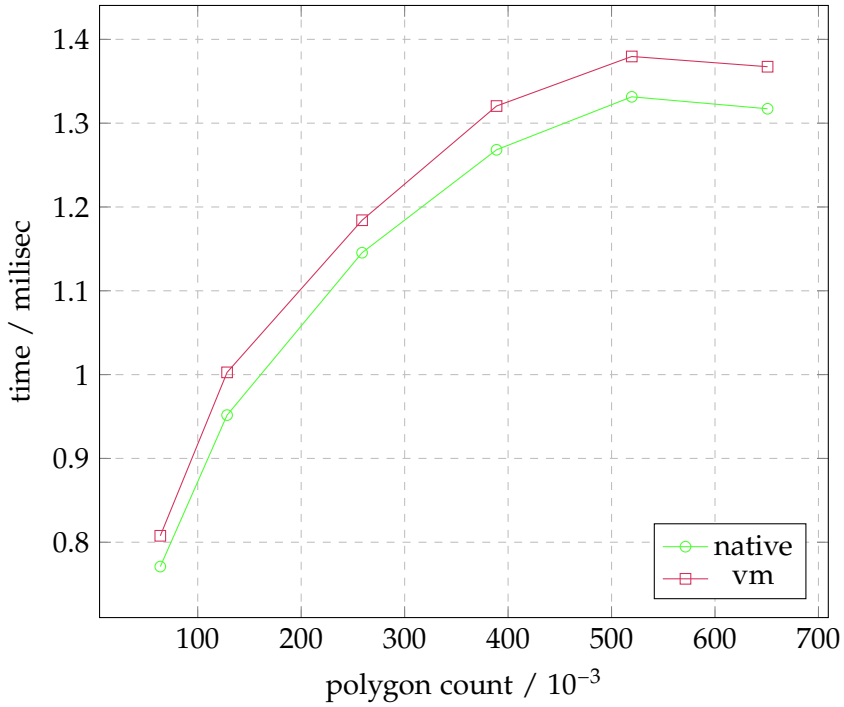


Figure 5.12: Average collision query time for the object Hand in native (○) and virtualization (□) environment using SIMDop CD algorithms for intel CPU (i7 7900x). The delta are similar across different object complexity.

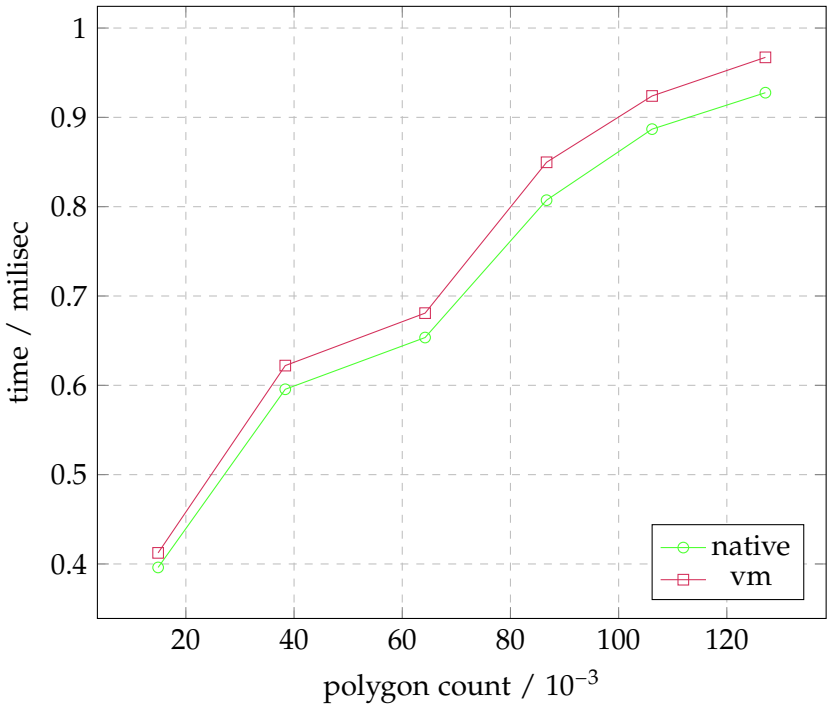


Figure 5.13: Average collision query time for the object Castle in native (○) and virtualization (□) environment using SIMDop CD algorithms for intel CPU (i7 7900x). The delta are similar across different object complexity.

for the public and can be accessed at URL: http://opencollbench.com.

We have presented openCollBench, a benchmarking architecture for CD and PQ 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. And at the same time provide a security guarantee while executing user-uploaded algorithms. 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. The security guarantee is done by shifting execution of user-uploaded algorithms in virtualization environment. 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 CD algorithm.

Our approach also offers interesting avenues for future work: for instance, currently, OpenCollBench is restricted to narrow phase CD and PQ 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.

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 benchmarking 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.

we also plan to collect quantitative and qualitative data about heatmap generated by our benchmarking server and objects uploaded by users (e.g. time taken to perform collision check by CD algorithms) in view of a larger and thorough evaluation of integrated CD algorithms.

We also plan to integrate more existing CD algorithms and extend generated heatmap based on other CD algorithms variables i.e., BV intersection test, triangles test, etc.

Our approach also offers interesting avenues for future work: for instance, by implementing the server endpoint as a REST endpoint, other services could also use the benchmarking server for example, to evaluate the proposed algorithm within a continuous integration pipeline when building an application. In this sense, a plugin for *integrated development environment (IDE)* such as *Visual Studio* would also be conceivable that allows the user to directly assess the effects of his changes to algorithms during development.

6

UNCERTAINTY IN SIMULATION-BASED ROBOT PLANNING

The current state of physically-based simulation development is well-known to produce plausible effects for physics simulation. From gaming to robotics, simulation provides clear benefits like a specific training environment for delicate tasks without potentially breaking the physical environment or the ability to envision the possible outcome of planned actions before execution, areas where simulation offers clear advantages. Moreover, the simulation can be easily reset in case something goes wrong.

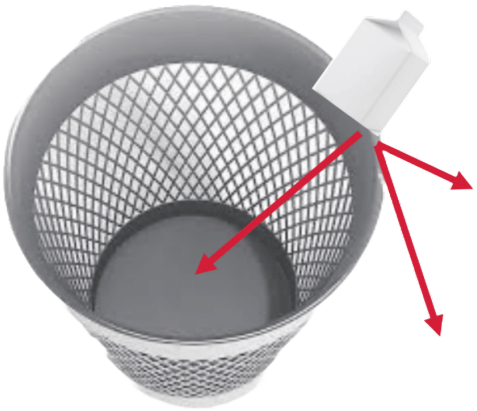
Despite the advantages, physics simulations are subject to uncertainty. In a general sense, uncertainty can arise from three sources, *simulation uncertainty*, lack of knowledge about the system being simulated, *input uncertainty*, uncertainty in the input data, and *structural uncertainty*, the choice of model and assumptions (McKay, Morrison, and Upton, 1999). All of these can lead to errors in the results of the simulation.

Among them, *input uncertainty* can be a major source of error in simulations, as even small input changes can lead to significant changes in the output. There are many error sources of *input uncertainty*, but one of the most common is measurement error. This can occur when measuring the position, velocity, or other properties of particles (Kenghagho et al., 2022). In such system, uncertainty plays a partial role in marginal Rao-Blackwellized Particle Filter (mRBPF) where numerous belief particles undergo simulation, weighting, and subsequent sampling according to their weights. Nevertheless, this approach may necessitate a substantial number of belief particles to accurately represent continuous physical parameters, particularly in scenarios involving collision and force dynamics crucial for simulating object behaviors. Hence, there is a need to minimize the particle count required.

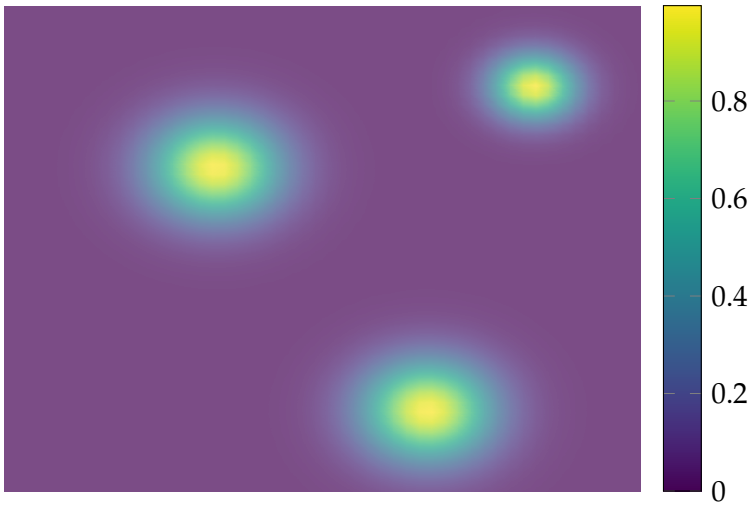
Another source of error could arise from imprecise understanding of the laws of physics. For example, in a simulation of the motion of the planets, the gravitational force between the planets is usually calculated using Newton's law of gravity. However, this law is only an approximation, and it is not known precisely how accurate it is.

There exist approaches in the literature that focus on *input uncertainty*. However, based on our knowledge, these methods mainly involve discrete sampling of input parameters (Kurniawati et al., 2011; Santolaria and GinéS, 2013), which require multiple simulation instances to conclude the simulation.

Our idea is to embed input uncertainty into existing CD algorithm, namely IST. The result of such simulation is a continuous distribution of final object states in the form of a heatmap (See Figure 6.1b). For example, if an object drops and hits the dustbin's edge, the object's final position may be inside or outside of the dustbin (See Figure 6.1a).



(a)



(b)

Figure 6.1: (a) Illustration of a milk box being dropped and hitting the edge of a dustbin. Red arrows point to possible final positions of the milk box, which could land inside or outside of the dustbin. (b) Heatmap showing the probabilistic distribution of the milk box's location after the simulation.

6.1 INNER SPHERE TREE FOR GEOMETRY WITH UNCERTAIN PROPERTIES

Assuming objects are rigid bodies, the size of IST spheres are fixed. Therefore, the input uncertainty, e. g., the position of objects, could be represented as a continuous probability distribution (i. e., Gaussian) by replacing the center or edge of spheres with a probability distribution.

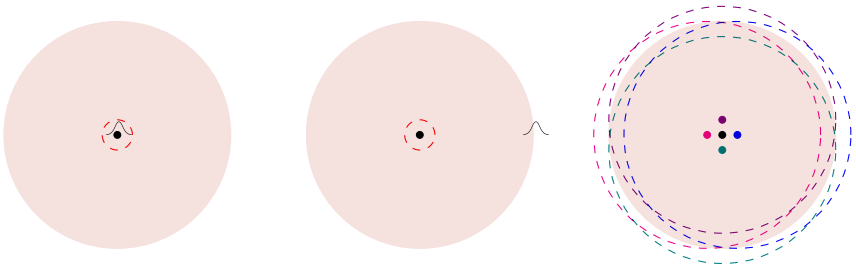


Figure 6.2: Sphere with position uncertainty represented with a probability distribution on (left) center and the (center) edge of spheres. (Right) Possible sphere positions.

Since continuous distribution is infinite, we need to limit distribution according to uncertainty confidence level, e. g., based on the three-sigma rule of thumb, to perform an overlap test between spheres. Limiting the distribution will allow us to have a second radius based on position uncertainty (See Figure 6.2). The Gaussian distribution depends on two variables, μ, and σ (See Equation 6.1). In our case, μ is the center or edge of spheres, and σ defines uncertainty confidence.

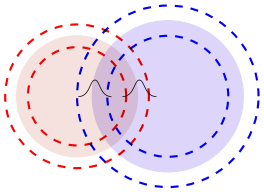
$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{6.1}$$

One way to avoid sampling for input uncertainty in CD algorithms, i. e., IST, is by representing the input parameter as a continuous probability function, thus being able to conclude simulation from one or several instances.

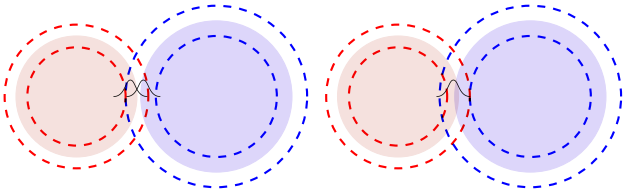
We modified the overlapping sphere test during simultaneous *inner bvh* traversal to check whether two objects with position uncertainty collide. With traditional IST, it is trivial to check, and cases either overlap or not. With input uncertainty, we identified three overlap cases (See algorithm 5). The modified overlap test starts by checking whether the second radius from both objects overlaps. We can stop the test if there is no overlap (See Figure 6.3c). Otherwise, we check the radius for overlap. If overlap, we can stop the test (See Figure 6.3a). Otherwise, there exists the possibility for overlap to some extent (See Figure 6.3b).

$$X_1, X_2 = \frac{\mu_1\sigma_2^2 - \mu_2\sigma_1^2 \pm \sigma_1\sigma_2\left[(\mu_1-\mu_2)^2 + (\sigma_2^2-\sigma_1^2)\log_e\frac{\sigma_2^2}{\sigma_1^2}\right]^{1/2}}{\sigma_2^2-\sigma_1^2} \tag{6.2}$$

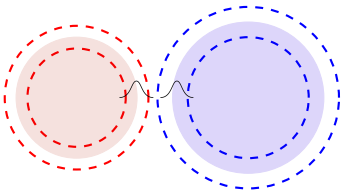
Once the traversal ends, we need to determine the probability distribution intersection for every pair of overlapping spheres to compute forces. This is necessary because the resulting force of one pair of overlapping spheres is not only pointed in one direction as in classic IST (See Figure 6.5a). Instead, the force points to several directions



(a) case 1: objects overlap



(b) case 2: possibility of objects overlap



(c) case3: objects do not overlap

Figure 6.3: Overlap cases during simultaneous *inner bvh* traversal between pair of objects with position uncertainty.

Algorithm 5: overlap(sphereNode *a*, sphereNode *b*)

if *dist(a_center, b_center)* >= *a_radius_uncertainty_max* + *b_radius_uncertainty_max* **then**
    `// case 3: objects do not overlap`
    return no_overlap
else if *dist(a_center, b_center)* <= *a_radius_uncertainty_min* + *b_radius_uncertainty_min* **then**
    `// case 1: objects overlap`
    return overlap
else
    `// case 2: possibility of objects overlap`
    return overlap_probability

due to position uncertainty and is represented using a probability distribution (See Figure 6.5b).

We could analytically determine the intersection of two unequal probability distributions (Inman and Jr, 1989). For unequal distributions, there exist intersections at two points (X_1, X_2) (See Equation 6.2).

For the equal case, the distributions are intersecting at a single point equal to $\frac{(\mu_1-\mu_2)}{2}$. The result of such an intersection is another probability distribution as well.

By summing the restitution force from all overlapping pairs as in classic IST, we can calculate the force direction for the next simulation step. This includes its probability representation. During simulation, the sum of the probability distribution for force could be too wide (See Figure 6.4). In this case, splitting force into two or several directions makes sense (See Figure 6.6). Depending on the application, a single simulation instance could develop into two or several instances.

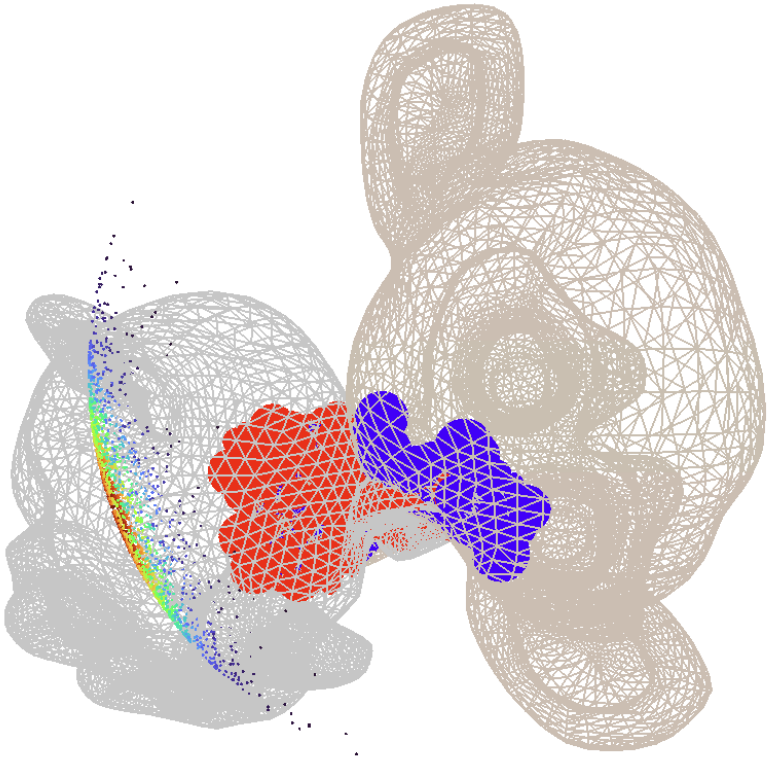
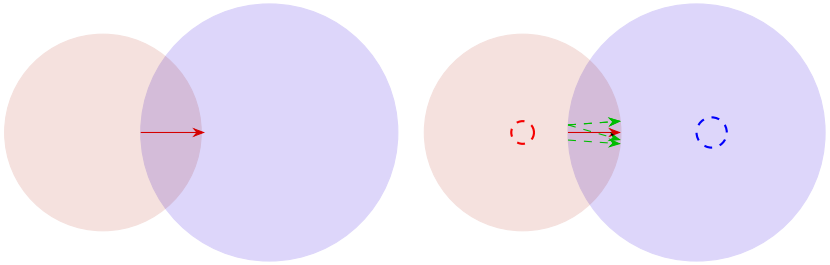


Figure 6.4: Resulting restitution force with the wide probability distribution (large σ)

6.2 PHYSICS SIMULATION WITH UNCERTAIN PROPERTIES

During simulation, uncertainty from a single step will affect the next simulation step. Therefore, the result of a single simulation step must be summed with the uncertainty of the next simulation step. It is well known that the probability distribution of the sum of two or more



(a) (b)

Figure 6.5: (a) Computation of forces using IST, and (b) the resulting force changes direction based on the probability distribution.

Algorithm 6: overlap(sphereNode *a*, sphereNode *b*)

```
if dist(a_center, b_center) >= a_radius_uncertainty_max +
  b_radius_uncertainty_max then
    // case 3: objects do not overlap
    return no_overlap
else if dist(a_center, b_center ) <= a_radius_uncertainty_min +
  b_radius_uncertainty_min then
    // case 1: objects overlap
    return overlap
else
    // case 2: possibility of objects overlap
    return overlap_probability
```

independent random variables is the convolution of their distributions, defined by:

$$h(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt. \tag{6.3}$$

The final result of such simulation can be seen in Figure 6.1b. For example, we simulated a dropped milk box into the edge of a dustbin. In this case, with the position uncertainty applied to simulation, the input is no more a single position of the object before its free-fall but rather a probabilistic distribution of its position.

The simulation output will be a probabilistic distribution of its location when it finishes the fall. The final object location, for instance, can be represented using a heatmap. This approach considerably reduces the number of simulation instances representing such distribution.

6.3 CONCLUSION AND FUTURE WORK

In summary, we have presented a novel method to address input uncertainty for physics simulation, using an example of position uncertainty. Our approach embeds uncertainty directly into the acceleration data structure used by CD, namely IST, which is capable of calculating

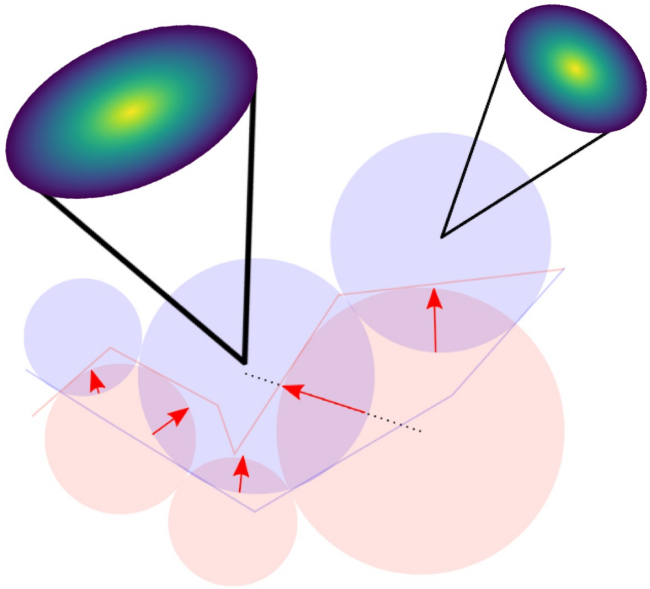


Figure 6.6: Elementary forces between two objects, a milk box (blue) and a dustbin (red), during a single simulation step are depicted in Figure 6.1a. Both objects are filled with spheres using IST algorithms. These forces are calculated based on collisions between spheres, resulting in directional changes embedded with a probability distribution.

collision and force needed for physics simulation. The result of the simulation is a continuous distribution of final object states in the form of a heatmap, considerably reducing the number of simulation instances needed to conclude the simulation.

Furthermore, our algorithm offers numerous other starting points for future work. For instance, in random sampling path-planning in non-static environments, our method can significantly reduce the number of samples needed. Adding additional variables such as a time limit or confidence level as in Klein and Zachmann, 2003 would be an exciting addition for future work. This addition would be particularly helpful in time-critical conditions where the simulation needs to conclude rapidly. Moreover, other input uncertainties, such as friction or mass, could be considered.

7 APPLICATIONS

7.1 COLLISION DETECION FOR GRASP TYPE DETECTION

Improvements in grasping algorithms and hand tracking methods now allow for natural interaction in virtual environments. This opens up possibilities for conducting experiments in Virtual Reality (VR), where participants can engage with virtual objects in a lifelike manner. Conducting experiments in VR makes it easier to gather information about the causal and intuitive physics of the environment or contact surfaces during grasping, compared to real-life setups. Additionally, repeating experiments with the same setup becomes more convenient in VR.

The data collected from these VR experiments can offer valuable insights into human grasping behaviors. This knowledge can be applied in various fields, such as robotic grasping, where the amount of data required to learn grasping tasks may surpass what can be feasibly provided with reasonable effort (Levine et al., 2018). Other fields, including prosthetics, or scenarios where only simple input devices are available for grasping in VR, can also benefit from these advancements.

Analyzing human grasping is a complex task influenced by various factors. To comprehend human grasping behavior, it's crucial to examine interactions with diverse objects, understanding where and how objects are touched and held. What may seem intuitive for humans could poses a significant challenge in robotics, i. e., how does a robot discern the distinct touch required for a cup compared to a cereal box? Furthermore, when grasping an object, determining the appropriate force to apply is another intricate aspect. VR provides an ideal platform for investigating these complexities by easily incorporating different objects and manipulation scenarios.

This Section, we introduce two steps to detect graps type based on GRASP taxonomy (Feix et al., 2015) during interaction within VR.

CONTACT POINTS AND FORCES DETECTION For detecting contact points and force, we opted for IST (Weller and Zachmann, 2011). IST not only allows us to identify which fingers are active but also provides information on contact points, penetration depths, and minimal distances.

The concept behind IST involves densely populating objects (in our case, the fingers and the kitchen item) with non-overlapping spheres Figure 7.1 and constructing a tree hierarchy based on this arrangement.

Figure 7.1: Hand filled with Spheres.

To fill the objects with spheres, we employed ProtoSphere (Weller and Zachmann, 2010), a GPU-assisted algorithm designed to pack a mesh with spheres.

The first step focuses on obtaining contact points and forces between the object and the hand. Contact points offer valuable information for visualizing grasp postures and can be beneficial for replicating grasps. These points may also serve as training data for robotic grasping. The method utilizes approach by Rosskamp et al., 2021 for generating heatmap for contact points and forces (forcemap) visualization (See Figure 7.2).

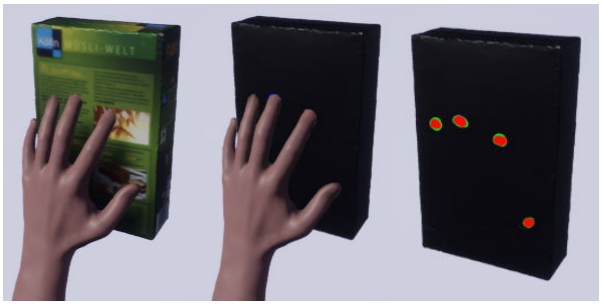


Figure 7.2: The hand grasping the object on the left, the texture for the forcemap in the middle and the final forcemap on the right.

An improvement involves defining a grid on the object, where each cell represents a potential grasping point. Colors are assigned based on how frequently a grid cell is in contact with a specific finger. Additionally, a force map is incorporated to illustrate the strength of the grasp at each point. Adequate force application is crucial for holding objects without causing them to fall or break. These heatmaps offer direct insights into where and how forcefully one should touch specific objects, potentially aiding in training robots to handle objects correctly.

GRASP TYPE DETECTION The second step focuses on discriminating between different grasp types, a vital aspect of human grasp behavior analysis. Recognizing that different objects require distinct

handling for proper treatment, the application employs a combination of collision detection and rotation angle analysis. This enables the discrimination of over 33 grasp types based on GRASP taxonomy (See Figure 7.3), facilitating a more nuanced understanding of human grasping behaviors. We opted out using decision tree based on interpretation of the GRASP taxonomy. In general, we group the grasps by which fingers are in use and how similarly the hand is shaped. In contrast to the taxonomic approach we disgard the thumb position as a cluster, as well as the hand's opposition type. Therefore, we don't keep exactly the same groupings as the taxonomy, but most grasp categories still remain collectively. While the taxonomy contained 17 grasp categories, our algorithm distinguishes between 6 active finger combinations (see Figure 7.3).

The first step of the Grasp Type Detection is finding which finger is touching the object using collision detection. The decision tree in Figure 7.3 is used for further discrimination due to the earlier introduced collision detection or rotation angle analysing functions.

While for grasp 15 *Fixed Hook* it is enough to detect it's unique finger combination (everyone except the thumb), the more similar grasps need a more detailed differentiation. To find grasp 22 *Parallel Extension*, all five fingers much be touching the object entirely. If the index finger is not straight, then the straightness of all the other opposing fingers gets detected. All straight fingers would then already indicate grasp 22, in contrast to grasp 17 *Index Finger Extension*, where middle, ring and pinky finger are facing towards the palm.

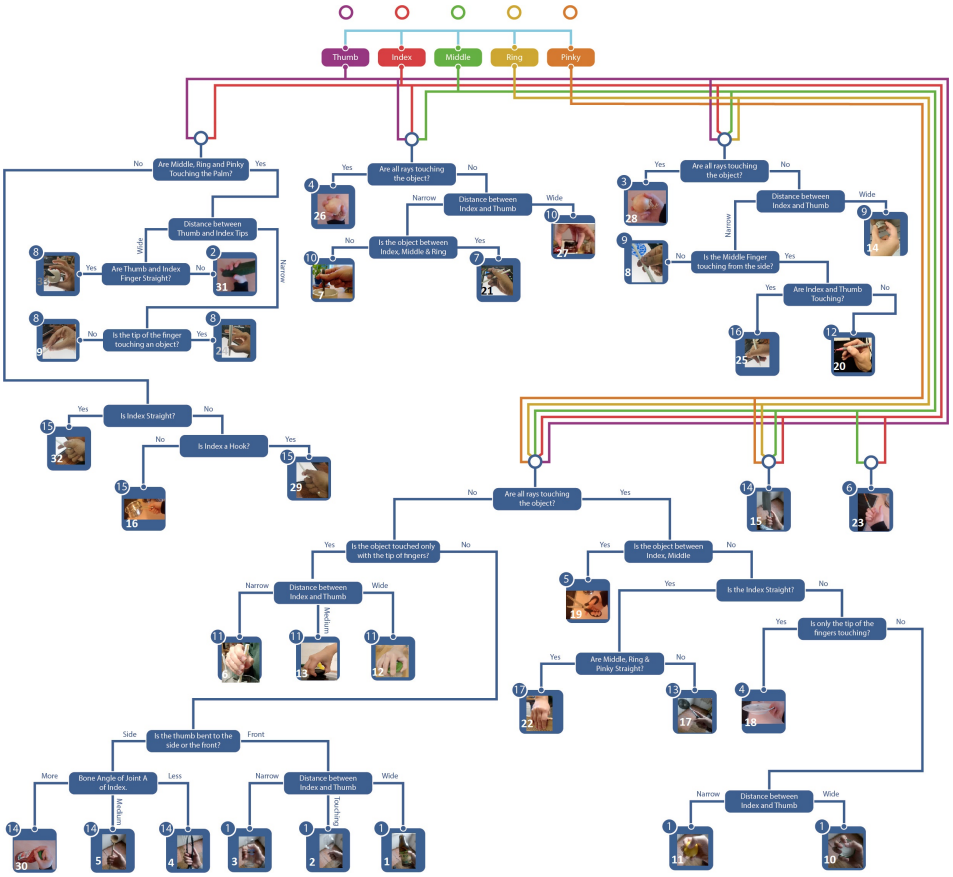


Figure 7.3: Decision tree for automatic grasp type detection based on GRASP Taxonomy for kitchen environment.

8 DISCUSSION AND CONCLUSION

As outlined in the research questions, this dissertation addresses fundamental challenges in physics simulation for robot planning, namely CD and uncertainty arising from sensing and action noise. To reach these objectives, we tackled the following tasks:

Faster-Than-Realtime Rigid Body Collision Detection

We have presented a top-down construction method for BVH based on clustering algorithms from machine learning, namely BNG, combined with SIMD-based simultaneous BVH traversal. The results are able to outperform algorithms like VC by an order of magnitude.

Memory-Efficient Collision Detection

We have introduced the idea to reduce the memory footprint of CD algorithms. Our idea is based on a lossy compression method using quantization. We applied the idea to two existing algorithms, namely *Doptree* & *Boxtree*.

Although there exists lossless compression methods for floating point, they are not suitable for BVH traversal tasks, especially when considering traversal time, as they are too slow.

Our method introduces a trade-off between traversal time and memory footprint. It becomes beneficial in scenarios where minimizing the required Random Access Memory (RAM) for cost reasons is crucial or where performance is not a major concern, such as during mental simulations where multiple instances need to be launched in parallel.

Reproducible and Comparable Benchmarking

We have introduced the idea to achieve reproducible and comparable benchmarking results, realized in the context of collision detection and proximity algorithms. We proposed a benchmarking architecture that offers the benchmarking procedure as an open web service to the public. The goal is to make complicated and time-consuming benchmarking accessible to both expert and non-expert users, while providing security guarantees when executing user-uploaded algorithms.

We 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. Security guarantees

are provided by executing user-uploaded algorithms in a virtualization environment. Additionally, we presented an extension to sub-object accuracy for the analysis of benchmarking results. The idea is to use heatmaps to visualize information gathered by the benchmark, allowing users to identify critical parts of their objects and enabling a better understanding of the behavior and characteristics of particular collision detection algorithms. We 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.

Physics Simulation with Continuous Probabilistic Distribution of Final State

We have presented a novel method to address input uncertainty for physics simulation using an example of position uncertainty. Our approach embeds uncertainty directly into the acceleration data structure used by collision detection. The result of such simulations is a continuous probabilistic distribution of the final state, considerably reducing the number of simulation instances needed to conclude the simulation.

8.1 LIMITATIONS AND FUTURE WORK

With the overall research question answered, certain limitations apply that call for future work from which some will we discussed in the following chapter.

8.1.1 *Simulation of Deformable Components*

Achieving high-speed robotics simulations is a crucial aspect of robot planning, particularly when it comes to tasks like model predictive control and online policy inference. The ability to run simulations faster than real-time is essential for efficient and effective robot operations. However, the challenge arises when dealing with models that incorporate flexible or deformable components, as they introduce additional computational complexity.

Simulating scenarios with unstructured and dynamically changing environments, such as deformable terrains or fluid-solid interactions, further adds to the computational demands and simulation slowdown. These aspects are important for enabling robots to operate successfully in complex real-world scenarios beyond controlled environments (Yoshida et al., 2015). To manage simulation time effectively, compromises must be made based on specific objectives and the available computing power. This may involve reducing numerical algorithm iterations, using coarser collision detection meshes, opting for rigid

terrains instead of deformable ones, simplifying sensing approaches, employing rigid elements instead of compliant ones, or even working with simplified two-dimensional dynamics. Model reduction techniques can also be employed to streamline the simulation process. However, it is crucial to consider the trade-offs as the nature of these compromises can vary depending on the available computing power, which may differ significantly between offline and online simulation setups. Adapting to these constraints while striving for high-speed simulations is essential for achieving efficient and realistic robot planning.

8.1.2 *Proper Handling of Uncertainty*

Uncertainty poses a significant challenge in the operation of robots, arising from various sources such as friction, impact, contact, actuator noise, and complex environments. To ensure accurate and reliable robot behavior, it is crucial to consider and account for uncertainty in simulations.

Addressing uncertainty involves incorporating mechanisms during both the model generation and simulation phases. During model setup, methods should be employed to introduce and handle uncertainty, such as allowing variations in friction coefficients, accounting for imperfect component geometries, and considering delays in actuation forces. During simulation, it is essential to effectively handle the lack of smoothness in robot dynamics' solutions, addressing conditions like stick-slip behavior and impacts. Embracing a statistical perspective with confidence bounds provides insights into the reliability of simulation results. Adopting an uncertainty quantification mindset is crucial as robots often operate in environments with limited knowledge and inherent uncertainty.

By actively addressing uncertainty in model generation and simulation, robotic systems can enhance their ability to operate in real-world scenarios. Accounting for uncertainty enables robots to make informed decisions, adapt to varying conditions, and improve system robustness and reliability.

BIBLIOGRAPHY

Agarwal, Pankaj K. et al. (2001). "Box-Trees and R-Trees with near-Optimal Query Time." In: *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*. SCG '01. New York, NY, USA: Association for Computing Machinery, 124–133. ISBN: 158113357X. DOI: 10.1145/378583.378645. URL: https://doi.org/10.1145/378583.378645.

Baek, Donghoon et al. (2018). "Path Planning for Automation of Surgery Robot based on Probabilistic Roadmap and Reinforcement Learning." In: *2018 15th International Conference on Ubiquitous Robots (UR)*, pp. 342–347. DOI: 10.1109/URAI.2018.8441801.

Bakhshalipour, Mohammad, Maxim Likhachev, and Phillip B. Gibbons (2022). "RTRBench: A Benchmark Suite for Real-Time Robotics." In: *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 175–186. DOI: 10.1109/ISPASS55109.2022.00024.

Battaglia, Peter W., Jessica B. Hamrick, and Joshua B. Tenenbaum (2013). "Simulation as an engine of physical scene understanding." In: *Proceedings of the National Academy of Sciences* 110.45, pp. 18327–18332. DOI: 10.1073/pnas.1306572110. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1306572110. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1306572110.

Bauszat, Pablo, Martin Eisemann, and Marcus Magnor (2010). "The Minimal Bounding Volume Hierarchy." In: *Vision, Modeling, and Visualization (2010)*. Ed. by Reinhard Koch, Andreas Kolb, and Christof Rezk-Salama. The Eurographics Association. ISBN: 978-3-905673-79-1. DOI: 10.2312/PE/VMV/VMV10/227-234.

Benthin, Carsten et al. (2018). "Compressed-leaf bounding volume hierarchies." In: *Proceedings of the Conference on High-Performance Graphics*, pp. 1–4.

Bergen, Gino van den (Jan. 1998). "Efficient Collision Detection of Complex Deformable Models Using AABB Trees." In: *J. Graph. Tools* 2.4, 1–13. ISSN: 1086-7651. DOI: 10.1080/10867651.1997.10487480. URL: https://doi.org/10.1080/10867651.1997.10487480.

Blum, Christian, Alan F. T. Winfield, and Verena V. Hafner (2018). "Simulation-Based Internal Models for Safer Robots." In: *Frontiers in Robotics and AI* 4. ISSN: 2296-9144. DOI: 10.3389/frobt.2017.00074. URL: https://www.frontiersin.org/articles/10.3389/frobt.2017.00074.

Bongard, Josh, Victor Zykov, and Hod Lipson (2006). "Resilient machines through continuous self-modeling." In: *Science* 314.5802, pp. 1118–1121.

Bonneel, Nicolas et al. (Aug. 2020). "Code Replicability in Computer Graphics." In: *ACM Trans. Graph.* 39.4. ISSN: 0730-0301. DOI: 10.1145/3386569.3392413. URL: https://doi.org/10.1145/3386569.3392413.

Bozcuoğlu, Asil Kaan and Michael Beetz (2017). "A cloud service for robotic mental simulations." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2653–2658. DOI: 10.1109/ICRA.2017.7989309.

Burgard, W. et al. (2005). "Coordinated multi-robot exploration." In: *IEEE Transactions on Robotics* 21.3, pp. 376–386. DOI: 10.1109/TRO.2004.839232.

Buss, Samuel R (2005). "Collision detection with relative screw motion." In: *The visual computer* 21, pp. 41–58.

Cao, Xiaoman et al. (2019). "RRT-based path planning for an intelligent litchi-picking manipulator." In: *Computers and Electronics in Agriculture* 156, pp. 105–118. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2018.10.031. URL: https://www.sciencedirect.com/science/article/pii/S0168169918303971.

Cavallo, Alberto et al. (2022). "Robotic Clerks: Autonomous Shelf Refilling." In: *Robotics for Intralogistics in Supermarkets and Retail Stores*. Ed. by Luigi Villani et al. Cham: Springer International Publishing, pp. 137–170. ISBN: 978-3-031-06078-6. DOI: 10.1007/978-3-031-06078-6_6. URL: https://doi.org/10.1007/978-3-031-06078-6_6.

Chen, Gang et al. (Dec. 2021). "Path Planning for Manipulators Based on an Improved Probabilistic Roadmap Method." In: *Robot. Comput.-Integr. Manuf.* 72.C. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2021.102196. URL: https://doi.org/10.1016/j.rcim.2021.102196.

Chinesta, Francisco et al. (2020). "Virtual, digital and hybrid twins: a new paradigm in data-based engineering and engineered data." In: *Archives of computational methods in engineering* 27, pp. 105–134.

Choi, Yi-King et al. (2009). "Continuous Collision Detection for Ellipsoids." In: *IEEE Transactions on Visualization and Computer Graphics* 15.2, pp. 311–325. DOI: 10.1109/TVCG.2008.80.

Choset, Howie (2000). "Coverage of known spaces: The boustrophedon cellular decomposition." In: *Autonomous Robots* 9, pp. 247–253.

Coming, Daniel S. and Oliver G. Staadt (2008). "Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection." In: *IEEE Transactions on Visualization and Computer Graphics* 14.1, pp. 1–12. DOI: 10.1109/TVCG.2007.70405.

Denny, Jory et al. (2020). "Dynamic Region-biased Rapidly-exploring Random Trees." In: *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Ed. by Ken Goldberg et al. Cham: Springer International Publishing, pp. 640–655. ISBN: 978-3-030-43089-4. DOI: 10.1007/978-3-030-43089-4_41. URL: https://doi.org/10.1007/978-3-030-43089-4_41.

Drepper, Ulrich (2007). "What every programmer should know about memory." In: *Red Hat, Inc* 11.2007, p. 2007.

Echeverria, Gilberto et al. (2011). "Modular open robots simulation engine: Morse." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, pp. 46–51.

Eckstein, Jens and Elmar Schömer (1999). "Dynamic collision detection in virtual reality applications." In: *Proc. The 7-th Int'l Conf. in Central Europe on Comp. Graphics, Vis. and Interactive Digital Media'99 (WSCG'99)*. Citeseer, pp. 71–78.

Feix, Thomas et al. (2015). "The grasp taxonomy of human grasp types." In: *IEEE Transactions on human-machine systems* 46.1, pp. 66–77.

Fong, Terrence, Illah Nourbakhsh, and Kerstin Dautenhahn (2003). "A survey of socially interactive robots." In: *Robotics and autonomous systems* 42.3-4, pp. 143–166.

Goldsmith, Jeffrey and John Salmon (May 1987). "Automatic Creation of Object Hierarchies for Ray Tracing." In: *IEEE Comput. Graph. Appl.* 7.5, pp. 14–20. ISSN: 0272-1716. DOI: 10.1109/MCG.1987.276983. URL: http://dx.doi.org/10.1109/MCG.1987.276983.

Gottschalk, S., M. C. Lin, and D. Manocha (1996). "OBBTree: A Hierarchical Structure for Rapid Interference Detection." In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 171–180. ISBN: 0897917464. DOI: 10.1145/237170.237244. URL: https://doi.org/10.1145/237170.237244.

Groß, Torben (2022). "Continuous Collision Detection with Inner Sphere Trees." Master's Thesis. University of Bremen.

Haidu, Andrei and Michael Beetz (2021). "Automated acquisition of structured, semantic models of manipulation activities from human VR demonstration." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9460–9466. DOI: 10.1109/ICRA48506.2021.9562016.

Hammer, Barbara, Alexander Hasenfuss, and Thomas Villmann (2007). "Magnification control for batch neural gas." In: *Neurocomputing* 70.7. Advances in Computational Intelligence and Learning, pp. 1225–1234. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2006.10.147. URL: https://www.sciencedirect.com/science/article/pii/S0925231206004759.

Han, Sangchul et al. (2022). "Snake Robot Gripper Module for Search and Rescue in Narrow Spaces." In: *IEEE Robotics and Automation Letters* 7.2, pp. 1667–1673. DOI: 10.1109/LRA.2022.3140812.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

He, Liang et al. (2015). "Interactive Continuous Collision Detection for Topology Changing Models Using Dynamic Clustering." In: *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. i3D '15. New York, NY, USA: Association for Computing Machinery, 47–54. ISBN: 9781450333924. DOI: 10.1145/2699276.2699286. URL: https://doi.org/10.1145/2699276.2699286.

Held, Martin, James T. Klosowski, and Joseph S. B. Mitchell (1996). "Collision Detection for Fly-Throughs in Virtual Environments." In: *Proceedings of the Twelfth Annual Symposium on Computational Geometry*. SCG '96. New York, NY, USA: Association for Computing Machinery, 513–514. ISBN: 0897918045. DOI: 10.1145/237218.237428. URL: https://doi.org/10.1145/237218.237428.

Hesslow, Germund (2012). "The current status of the simulation theory of cognition." In: *Brain Research* 1428. The Cognitive Neuroscience of Thought, pp. 71–79. ISSN: 0006-8993. DOI: https://doi.org/10.1016/j.brainres.2011.06.026. URL: https://www.sciencedirect.com/science/article/pii/S0006899311011309.

Hu, Shuyu et al. (2022). "Underwater gas leak detection using an autonomous underwater vehicle (robotic fish)." In: *Process Safety and Environmental Protection* 167, pp. 89–96.

Hubbard, Philip M. (July 1996). "Approximating Polyhedra with Spheres for Time-Critical Collision Detection." In: *ACM Trans. Graph.* 15.3, 179–210. ISSN: 0730-0301. DOI: 10.1145/231731.231732. URL: https://doi.org/10.1145/231731.231732.

Huisman, Chantal and Helianthe Kort (2019). "Two-Year Use of Care Robot Zora in Dutch Nursing Homes: An Evaluation Study." In: *Healthcare* 7.1. ISSN: 2227-9032. DOI: 10.3390/healthcare7010031. URL: https://www.mdpi.com/2227-9032/7/1/31.

Inman, Henry F. and Edwin L. Bradley Jr (1989). "The overlapping coefficient as a measure of agreement between probability distributions and point estimation of the overlap of two normal densities." In: *Communications in Statistics - Theory and Methods* 18.10, pp. 3851–3874. DOI: 10.1080/03610928908830127. eprint: https://doi.org/10.1080/03610928908830127. URL: https://doi.org/10.1080/03610928908830127.

Johansson, Birger and Christian Balkenius (2006). "An experimental study of anticipation in simple robot navigation." In: *Workshop on anticipatory behavior in adaptive learning systems*. Springer, pp. 365–378.

– (2008). "Prediction time in anticipatory systems." In: *Workshop on Anticipatory Behavior in Adaptive Learning Systems*. Springer, pp. 283–300.

Kavraki, L.E. et al. (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." In: *IEEE Transactions on Robotics and Automation* 12.4, pp. 566–580. DOI: 10.1109/70.508439.

Kenghagho, Franklin K. et al. (2022). "NaivPhys4RP - Towards Human-like Robot Perception "Physical Reasoning based on Embodied Probabilistic Simulation"." In: *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pp. 815–822. DOI: 10.1109/Humanoids53995.2022.10000153.

Kim, Duksu et al. (2009a). "HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs." In: *Computer Graphics Forum*. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2009.01556.x.

Kim, Tae-Joon et al. (2009b). "RACBVHs: Random-Accessible Compressed Bounding Volume Hierarchies." In: *SIGGRAPH 2009: Talks*. SIGGRAPH '09. New York, NY, USA: Association for Computing Machinery. ISBN: 9781605588346. DOI: 10.1145/1597990.1598036. URL: https://doi.org/10.1145/1597990.1598036.

Klein, Jan and Gabriel Zachmann (2003). "ADB-Trees: Controlling the Error of Time-Critical Collision Detection." In: *VMV*, pp. 37–45.

Klosowski, James Thomas (1998). "Efficient Collision Detection for Interactive Three-Dimensional Graphics and Virtual Environments." AAI9904092. PhD thesis. USA. ISBN: 0599014725.

Koenig, N. and A. Howard (2004). "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

Konijn, Elly A et al. (2022). "Social robots for (second) language learning in (migrant) primary school children." In: *International Journal of Social Robotics*, pp. 1–17.

Koskela, Matias et al. (2015). "Using half-precision floatingpoint numbers for storing bounding volume hierarchies." In: *Proceedings of the 32nd Computer Graphics International Conference*.

Krishnan, Shankar et al. (1998). "Rapid and accurate contact determination between spline models using ShellTrees." In: *Computer Graphics Forum*. Vol. 17. 3. Wiley Online Library, pp. 315–326.

Kunze, Lars and Michael Beetz (2017). "Envisioning the qualitative effects of robot manipulation actions using simulation-based projections." In: *Artificial Intelligence* 247. Special Issue on AI and Robotics, pp. 352–380. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0004370214001544.

Kurniawati, Hanna et al. (2011). "Motion planning under uncertainty for robotic tasks with long time horizons." In: *The International Journal of Robotics Research* 30.3, pp. 308–323.

Kyrarini, Maria et al. (2021). "A Survey of Robots in Healthcare." In: *Technologies* 9.1. ISSN: 2227-7080. DOI: 10.3390/technologies9010008. URL: https://www.mdpi.com/2227-7080/9/1/8.

LaValle, Steven (1998). "Rapidly-exploring random trees: A new tool for path planning." In: *Research Report 9811*.

Larsen, E. et al. (2000). "Fast distance queries with rectangular swept sphere volumes." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 4, 3719–3726 vol.4. DOI: 10.1109/ROBOT.2000.845311.

Lauterbach, Christian, Sung-Eui Yoon, and Dinesh Manocha (2007). "Ray-Strips: A Compact Mesh Representation for Interactive Ray Tracing." In: *2007 IEEE Symposium on Interactive Ray Tracing*, pp. 19–26. DOI: 10.1109/RT.2007.4342586.

Lauterbach, Christian et al. (2009). "Fast BVH Construction on GPUs." In: *Computer Graphics Forum* 28.2, pp. 375–384. URL: http://dblp.uni-trier.de/db/journals/cgf/cgf28.html\#LauterbachGSLM09.

Leutenegger, Scott T., Jeffrey M. Edgington, and Mario A. Lopez (1997). *STR: A simple and efficient algorithm for R-tree packing*. Tech. rep. Institute for Computer Applications in Science and Engineering (ICASE).

Levine, Sergey et al. (2018). "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *The International journal of robotics research* 37.4-5, pp. 421–436.

Lian, Shiqi et al. (2018). "Dadu-P: A Scalable Accelerator for Robot Motion Planning in a Dynamic Environment." In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6. DOI: 10.1109/DAC.2018.8465785.

Lindqvist, Björn et al. (2022). "Multimodality robotic systems: Integrated combined legged-aerial mobility for subterranean search-and-rescue." In: *Robotics and Autonomous Systems* 154, p. 104134. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2022.104134. URL: https://www.sciencedirect.com/science/article/pii/S0921889022000756.

Lubiw, Anna, Jack Snoeyink, and Hamideh Vosoughpour (2017). "Visibility graphs, dismantlability, and the cops and robbers game." In: *Computational Geometry* 66, pp. 14–27. ISSN: 0925-7721. DOI: https://doi.org/10.1016/j.comgeo.2017.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0925772117300688.

Mania, Patrick and Michael Beetz (2019). "A Framework for Self-Training Perceptual Agents in Simulated Photorealistic Environments." In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4396–4402. DOI: 10.1109/ICRA.2019.8793474.

Martinetz, T.M., S.G. Berkovich, and K.J. Schulten (1993). "'Neural-gas' network for vector quantization and its application to time-series prediction." In: *IEEE Transactions on Neural Networks* 4.4, pp. 558–569. DOI: 10.1109/72.238311.

McKay, Michael D, John D Morrison, and Stephen C Upton (1999). "Evaluating prediction uncertainty in simulation models." In: *Computer Physics Communications* 117.1-2, pp. 44–51.

Merkt, Wolfgang, Vladimir Ivan, and Sethu Vijayakumar (2019). "Continuous-Time Collision Avoidance for Trajectory Optimization in Dynamic Environments." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7248–7255. DOI: 10.1109/IROS40897.2019.8967641.

Metcalfe, J. S. et al. (2017). "Building a framework to manage trust in automation." In: *Micro- and Nanotechnology Sensors, Systems, and Applications IX*. Ed. by Thomas George, Achyut K. Dutta, and M. Saif Islam. Vol. 10194. International Society for Optics and Photonics. SPIE, 101941U. DOI: 10.1117/12.2264245. URL: https://doi.org/10.1117/12.2264245.

Michel, Olivier (2004). "Cyberbotics ltd. webots™: professional mobile robot simulation." In: *International Journal of Advanced Robotic Systems* 1.1, p. 5.

Millard, Alan G, Jon Timmis, and Alan FT Winfield (2014a). "Run-time detection of faults in autonomous mobile robots based on the comparison of simulated and real robot behaviour." In: *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 3720–3725.

– (2014b). "Towards exogenous fault detection in swarm robotic systems." In: *Towards Autonomous Robotic Systems: 14th Annual Conference, TAROS 2013, Oxford, UK, August 28–30, 2013, Revised Selected Papers 14*. Springer, pp. 429–430.

Mirtich, Brian (2000). "Timewarp Rigid Body Simulation." In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., 193–200. ISBN: 1581132085. DOI: 10.1145/344779.344866. URL: https://doi.org/10.1145/344779.344866.

Murray, Sean et al. (2016). "Robot Motion Planning on a Chip." In: *Robotics: Science and Systems*. Vol. 6.

Murray, Sean et al. (2019). "A Programmable Architecture for Robot Motion Planning Acceleration." In: *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. Vol. 2160-052X, pp. 185–188. DOI: 10.1109/ASAP.2019.0004.

Niu, Hanlin et al. (2019). "Voronoi-Visibility Roadmap-based Path Planning Algorithm for Unmanned Surface Vehicles." In: *The Journal of Navigation* 72.4, 850–874. DOI: 10.1017/S0373463318001005.

Pasalidou, Christina, Nikolaos Fachantidis, and Efthymia Koiou (2023). "Using Augmented Reality and a Social Robot to Teach Geography in Primary School." In: *Learning and Collaboration Technologies*. Ed. by Panayiotis Zaphiris and Andri Ioannou. Cham: Springer Nature Switzerland, pp. 371–385. ISBN: 978-3-031-34550-0.

Ponamgi, Madhav, Dinesh Manocha, and Ming C. Lin (1995). "Incremental algorithms for collision detection between solid models." In: *Proceedings of the Third ACM Symposium on Solid Modeling and*

Applications. SMA '95. Salt Lake City, Utah, USA: Association for Computing Machinery, 293–304. ISBN: 0897916727. DOI: 10.1145/218013.218076. URL: https://doi.org/10.1145/218013.218076.

Püschel, A, C Schafmayer, and J Groß (2022). "Robot-assisted techniques in vascular and endovascular surgery." In: *Langenbeck's archives of surgery* 407.5, pp. 1789–1795.

Redon, S., M. C. Lin, and D. Manocha (2004). "Fast Continuous Collision Detection for Articulated Models." In: *Solid Modeling*. Ed. by Gershon Elber, Nicholas Patrikalakis, and Pere Brunet. The Eurographics Association. ISBN: 3-905673-55-X. DOI: 10.2312/sm.20041385.

Redon, Stéphane, Abderrahmane Kheddar, and Sabine Coquillart (2002). "Fast Continuous Collision Detection between Rigid Bodies." In: *Computer Graphics Forum* 21.3, pp. 279–287. DOI: https://doi.org/10.1111/1467-8659.t01-1-00587. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.t01-1-00587. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.t01-1-00587.

Reggiani, M., M. Mazzoli, and S. Caselli (2002). "An experimental evaluation of collision detection packages for robot motion planning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 3, 2329–2334 vol.3. DOI: 10.1109/IRDS.2002.1041615.

Richter-Klug, Jesse et al. (2022). "Improving Object Pose Estimation by Fusion With a Multimodal Prior – Utilizing Uncertainty-Based CNN Pipelines for Robotics." In: *IEEE Robotics and Automation Letters* 7.2, pp. 2282–2288. DOI: 10.1109/LRA.2022.3140450.

Rosskamp, Janis et al. (2021). "UnrealHaptics: Plugins for Advanced VR Interactions in Modern Game Engines." In: *Frontiers in Virtual Reality* 2, p. 32. ISSN: 2673-4192. DOI: 10.3389/frvir.2021.640470. URL: https://www.frontiersin.org/article/10.3389/frvir.2021.640470.

Roussopoulos, Nick and Daniel Leifker (1985). "Direct spatial search on pictorial databases using packed R-trees." In: *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*. SIGMOD '85. Austin, Texas, United States: ACM, pp. 17–31. ISBN: 0-89791-160-1. DOI: 10.1145/318898.318900. URL: http://doi.acm.org/10.1145/318898.318900.

Santiago, Robert Martin C. et al. (2017). "Path planning for mobile robots using genetic algorithm and probabilistic roadmap." In: *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–5. DOI: 10.1109/HNICEM.2017.8269498.

Santolaria, Jorge and Manuel GinéS (2013). "Uncertainty estimation in robot kinematic calibration." In: *Robotics and Computer-Integrated Manufacturing* 29.2, pp. 370–384.

Schwochow, Alexander (2021). "Compression of the BoxTree data structure." Master's Thesis. University of Bremen.

Solenthaler, B. and R. Pajarola (2009). "Predictive-corrective incompressible SPH." In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH '09. New Orleans, Louisiana: Association for Computing Machinery. ISBN: 9781605587264. DOI: 10.1145/1576246.1531346. URL: https://doi.org/10.1145/1576246.1531346.

Taeubig, Holger and Udo Frese (2012). "A New Library for Real-time Continuous Collision Detection." In: *ROBOTIK 2012; 7th German Conference on Robotics*, pp. 1–5.

Tan, Toni, Rene 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. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450381697. DOI: 10.1145/3424616.3424712. URL: https://doi.org/10.1145/3424616.3424712.

– (2022). "A Framework for Safe Execution of User-Uploaded Algorithms." In: *Proceedings of the 27th International Conference on 3D Web Technology*. Web3D '22. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450399142. DOI: 10.1145/3564533.3564560. URL: https://doi.org/10.1145/3564533.3564560.

Tan, Toni, René Weller, and Gabriel Zachmann (2019). "SIMDop: SIMD optimized Bounding Volume Hierarchies for Collision Detection." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7256–7263. DOI: 10.1109/IROS40897.2019.8968492.

Tan, Toni et al. (2021). "Grasping for reality-How can we improve the digital representation of human grasp behaviour?" In: *GI VR/AR Workshop*. Gesellschaft für Informatik eV.

Tan, Ying and Zhong yang Zheng (2013). "Research Advance in Swarm Robotics." In: *Defence Technology* 9.1, pp. 18–39. ISSN: 2214-9147. DOI: https://doi.org/10.1016/j.dt.2013.03.001. URL: https://www.sciencedirect.com/science/article/pii/S221491471300024X.

Tang, Min et al. (2008). "Interactive Continuous Collision Detection between Deformable Models Using Connectivity-Based Culling." In: *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*. SPM '08. New York, NY, USA: Association for Computing Machinery, 25–36. ISBN: 9781605581064. DOI: 10.1145/1364901.1364908. URL: https://doi.org/10.1145/1364901.1364908.

Toni, Toni, Rene Weller, and Gabriel Zachmann (2017). "SIMD Optimized Bounding Volume Hierarchies for Collision Detection." In: *GI VR/AR Workshop*. Gesellschaft für Informatik eV.

Trenkel, Sven, René Weller, and Gabriel Zachmann (Jan. 2007). "A Benchmarking Suite for Static Collision Detection Algorithms." In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Ed. by Václav Skala. Plzen,

Czech Republic: Union Agency. URL: http://cg.in.tu-clausthal.de/research/colldet_benchmark.

Vaidyanathan, Karthikeyan, Tomas Akenine-Möller, and Marco Salvi (2016). "Watertight ray traversal with reduced precision." In: *High Performance Graphics*, pp. 33–40.

Vaughan, Richard T and B Gerkey (2007). "Really reusable robot code and the player/stage project." In: *Software Engineering for Experimental Robotics. Springer*.

Viitanen, T. et al. (July 2017). "Fast Hardware Construction and Refitting of Quantized Bounding Volume Hierarchies." In: *Comput. Graph. Forum* 36.4, 167–178. ISSN: 0167-7055. DOI: 10.1111/cgf.13233. URL: https://doi.org/10.1111/cgf.13233.

Wald, Ingo (2007). "On fast Construction of SAH-based Bounding Volume Hierarchies." In: *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*. RT '07. Washington, DC, USA: IEEE Computer Society, pp. 33–40. ISBN: 978-1-4244-1629-5. DOI: 10.1109/RT.2007.4342588. URL: http://dx.doi.org/10.1109/RT.2007.4342588.

Wald, Ingo and Vlastimil Havran (2006). "On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)." In: *Symposium on Interactive Ray Tracing* 0, pp. 61–69. DOI: http://doi.ieeecomputersociety.org/10.1109/RT.2006.280216.

Weller, René and Gabriel Zachmann (Dec. 2010). "ProtoSphere: A GPU-Assisted Prototype-Guided Sphere Packing Algorithm for Arbitrary Objects." In: *ACM SIGGRAPH ASIA 2010 Sketches*. Seoul, Republic of Korea: ACM, 8:1–8:2. ISBN: 978-1-4503-0523-5. DOI: http://doi.acm.org/10.1145/1899950.1899958. URL: http://cg.in.tu-clausthal.de/research/protosphere.

Weller, Rene and Gabriel Zachmann (2011). "Inner Sphere Trees and Their Application to Collision Detection." In: *Virtual realities*. Springer, pp. 181–201.

Weller, Rene et al. (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*, pp. 63–70.

Weller, René et al. (2014). "Massively Parallel Batch Neural Gas for Bounding Volume Hierarchy Construction." In: *Workshop on Virtual Reality Interaction and Physical Simulation*. Ed. by Jan Bender et al. The Eurographics Association. ISBN: 978-3-905674-71-2. DOI: 10.2312/vriphys.20141219.

Wolper, Joshuah et al. (July 2019). "CD-MPM: continuum damage material point methods for dynamic fracture animation." In: *ACM Trans. Graph.* 38.4. ISSN: 0730-0301. DOI: 10.1145/3306346.3322949. URL: https://doi.org/10.1145/3306346.3322949.

Robotic Fish Enabled Offshore Pipeline Inspection (May 2023). Vol. Day 4 Thu, May 04, 2023. OTC Offshore Technology Conference, D041S055R004. DOI: 10.4043/32427-MS. eprint: https://onepetro.org/OTCONF/

proceedings-pdf/23OTC/4-23OTC/D041S055R004/3103458/otc-32427-ms.pdf. URL: https://doi.org/10.4043/32427-MS.

Yoshida, Eiichi et al. (2015). "Simulation-based optimal motion planning for deformable object." In: *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pp. 1–6. DOI: 10.1109/ARSO.2015.7428219.

Zachmann, G. (1998). "Rapid collision detection by dynamically aligned DOP-trees." In: *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180)*, pp. 90–97. DOI: 10.1109/VRAIS.1998.658428.

Zachmann, Gabriel (1995). "The boxtree: Exact and fast collision detection of arbitrary polyhedra." In: *First Workshop on Simulation and Interaction in Virtual Environments (SIVE 95)*. Citeseer.

– (May 2000). "Virtual Reality in Assembly Simulation – Collision Detection, Simulation Algorithms, and Interaction Techniques." Dissertation. Darmstadt University of Technology, Germany. ISBN: ISBN 3-8167-5628-X.

Zachmann, Gabriel and Elmar Langetepe (July 2003). "Geometric Data Structures for Computer Graphics." In: *Proc. of ACM SIGGRAPH*. ACM Transactions of Graphics. URL: http://www.gabrielzachmann.org/.

Zhang, Xinyu, Minkyoung Lee, and Young J Kim (2006). "Interactive continuous collision detection for non-convex polyhedra." In: *The Visual Computer* 22, pp. 749–760.

Zhang, Xinyu et al. (July 2007). "Continuous Collision Detection for Articulated Models Using Taylor Models and Temporal Culling." In: *ACM Trans. Graph.* 26.3, 15–es. ISSN: 0730-0301. DOI: 10.1145/1276377.1276396. URL: https://doi.org/10.1145/1276377.1276396.

Zhao, Yang et al. (2022). "Remote vascular interventional surgery robotics: A literature review." In: *Quantitative Imaging in Medicine and Surgery* 12.4, p. 2552.

xkcd.com (2024). *Dependency Hell*. https://xkcd.com/1579/ [Accessed: 14-02-2024].

FUNDAMENTAL PUBLICATIONS

This dissertation is based on five fundamental publications, which encompass several contributions: faster-than-realtime collision detection, benchmarking as an online service, and the exploration of uncertain physics in research on simulation-based robot planning. In the following, the publications which solidify this dissertations are briefly summarized and the personal contribution according to the CRediT taxonomy[1] are stated.

F 1 SIMDOP: SIMD OPTIMIZED BOUNDING VOLUME HIERARCHIES FOR COLLISION DETECTION

Toni Tan, René Weller, and Gabriel Zachmann (2019). "SIMDop: SIMD optimized Bounding Volume Hierarchies for Collision Detection." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7256–7263. DOI: 10.1109/IROS40897.2019.8968492

Place in this dissertation: This publication focuses on investigating acceleration strategies, specifically in relation to Research Question 1, for the newly released 512-bit Advanced Vector Extensions (AVX-512). It further includes a comparison with commonly used algorithms for robot planning, such as PQP and VC. The key contribution of this work lies in the development of novel acceleration strategies tailored specifically for AVX-512.

Contribution to publication: Conceptualization, data curation, formal analysis, investigation, methodology development, software development, validation, visualization, and contribution to all sections of the manuscript (70%).

F 2 NAIVPHYS4RP - TOWARDS HUMAN-LIKE ROBOT PERCEPTION "PHYSICAL REASONING BASED ON EMBODIED PROBABILISTIC SIMULATION"

Franklin K. Kenghagho, Michael Neumann, Patrick Mania, Toni Tan, Feroz A. Siddiky, René Weller, Gabriel Zachmann, and Michael Beetz (2022). "NaivPhys4RP - Towards Human-like Robot Perception "Physical Reasoning based on Embodied Probabilistic Simulation"." In: *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pp. 815–822. DOI: 10.1109/Humanoids53995.2022.10000153

Place in this dissertation: This publication makes a valuable contribution by addressing the challenges associated with achieving realistic robot belief, particularly in relation to uncertain physical parameters

1 https://casrai.org/credit/

in simulations, as outlined in Research Question 3. In the context of belief propagation using mRBPF, one of the significant challenges lies in dealing with uncertainties surrounding physical parameters, such as friction, mass, or object position in the world. These uncertainties can have a significant impact on the accuracy of simulations and subsequent belief updates.

To address this challenge, the publication proposes a novel approach that embeds uncertainty directly into the simulation process, with a specific focus on CD tasks. By incorporating uncertainty into the simulation, the number of belief particles required to accurately represent the physical parameters can be reduced, particularly for continuous physical quantities. This approach has the potential to improve the efficiency and effectiveness of the mRBPF framework, enhancing the overall performance of robot belief systems.

Contribution to publication: My personal contribution to this research includes conceptualization on uncertain physics, data curation, formal analysis, investigation, methodology development, visualization, and contributing to the sections of the manuscript related to uncertain physics (20%).

F 3 OPENCOLLBENCH - BENCHMARKING OF COLLISION DETECTION & PROXIMITY QUERIES AS A WEB-SERVICE

Toni Tan, Rene 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. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450381697. DOI: 10.1145/3424616.3424712. URL: https://doi.org/10.1145/3424616.3424712

Place in this dissertation: This publication makes a contribution by addressing the design of a comparable and reproducible benchmark for CD and PQ tasks, as outlined in Research Question 2. The focus is on developing a benchmark that allows for fair evaluations of different algorithms in CD and PQ, taking into account various factors such as object shape, relative size, distance between objects, and distributions of geometric primitives. This is realized in form of benchmark as online service.

Moreover, the publication delves into the representation of benchmarking results to gain a comprehensive understanding of the strengths and weaknesses of different algorithms. By analyzing and interpreting the benchmarking results, the research aims to identify the factors that influence the performance variations of CD and PQ algorithms. This contribution contributes to the field by providing valuable insights into algorithm performance and facilitating future improvements in CD and PQ tasks.

Contribution to publication: Conceptualization, data curation, formal analysis, investigation, methodology development, software development, validation, visualization, and contribution to all sections of the manuscript (70%).

F 4 A FRAMEWORK FOR SAFE EXECUTION OF USER-UPLOADED ALGORITHMS

Toni Tan, Rene Weller, and Gabriel Zachmann (2022). "A Framework for Safe Execution of User-Uploaded Algorithms." In: *Proceedings of the 27th International Conference on 3D Web Technology*. Web3D '22. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450399142. DOI: 10.1145/3564533.3564560. URL: https://doi.org/10.1145/3564533.3564560

Place in this dissertation: This publication provides a contribution by addressing the challenge of offering the benchmark as an online service, as outlined in Research Question 2. One of the key challenges in this regard is the integration of new algorithms into the existing benchmarking tool while ensuring the security of the system. The primary concern is the potential risks associated with running unknown code, such as the possibility of Remote Code Execution (RCE).

To mitigate these security concerns, this publication focuses on implementing a solution that guarantees the security of the benchmarking process. This is achieved by shifting the execution of user-uploaded algorithms into a virtualization environment. By isolating the execution environment, it ensures that any potential risks posed by the uploaded code are contained within the virtualized environment, protecting the underlying system and maintaining the overall security of the benchmarking service.

This contribution not only enables the integration of new algorithms into the benchmarking tool but also ensures the safety and security of the online service, providing researchers with a reliable platform to evaluate and compare their algorithms in a secure manner.

Contribution to publication: Conceptualization, data curation, formal analysis, investigation, methodology development, software development, validation, visualization, and contribution to all sections of the manuscript (70%).

F 5 SIMD OPTIMIZED BOUNDING VOLUME HIERARCHIES FOR COLLISION DETECTION

Toni Toni, Rene Weller, and Gabriel Zachmann (2017). "SIMD Optimized Bounding Volume Hierarchies for Collision Detection." In: *GI VR/AR Workshop*. Gesellschaft für Informatik eV

Place in this dissertation: This publication delves into a seldom-explored aspect of CD by investigating the influence of BVH branching

factors. In relation to Research Question 1, this work contributes to an integrated approach for accelerating collision detection tasks by utilizing an appropriate branching factor based on the available SIMD register, in combination with BNG-based BVH.

Contribution to publication: Conceptualization, data curation, formal analysis, investigation, methodology development, software development, validation, visualization, and contribution to all sections of the manuscript (70%).

SUPPORTIVE PUBLICATIONS

These publications are related work which supplements the fundamental publications.

S 1 GRASPING FOR REALITY-HOW CAN WE IMPROVE THE DIGITAL REPRESENTATION OF HUMAN GRASP BEHAVIOUR?

Toni Tan, Janis Rosskamp, Rene Weller, and Gabriel Zachmann (2021). "Grasping for reality-How can we improve the digital representation of human grasp behaviour?" In: *GI VR/AR Workshop*. Gesellschaft für Informatik eV

Place in this dissertation: This publication explores the representation of human grasps in VR using heat- and forcemaps. The research focuses on recognizing different grasp types while objects are manipulated in a VR environment.

The application of this research offers a novel and valuable approach to representing hand grasps digitally, with potential applications across various domains. The findings can contribute to advancements in robotics and prosthetics, as understanding how humans grasp objects is crucial in developing more realistic and effective hand representations.

By providing a digital representation of hand grasps, this research opens up possibilities for enhancing virtual experiences, improving human-computer interactions, and facilitating the development of more advanced and immersive VR applications.

Contribution to publication: Conceptualization, project administration, contribution to all parts of the manuscript (40%).

APPENDIX

A 1 INTRINSICS CODE FOR SIMD-BASED SIMULTANEOUS BVH TRAVERSAL

A 1.1 *1 vs 16*

Algorithm 7: _m512 intersect(DOP a, DOP b1,...,b16)

```
_mm512 endResult
for i=0; i<k/2; i++ do
    _mm512 oriAL = _mm512_set1_ps(a[i])
    _mm512 oriBL = _mm512_set_ps(b1[i],...,b16[i])
    _mm512 resL = _mm512_cmp_ps(oriAL, oriBL, _CMP_LT_OS)
    _mm512 oriAH = _mm512_set1_ps(a[k/2+i])
    _mm512 oriBH = _mm512_set_ps(b1[k/2+i],...,b16[k/2+i])
    _mm512 resH = _mm512_cmp_ps(oriAH, oriBH, _CMP_GT_OS)
    _mm512 tempRes = _mm512_kor(resL,resH)
    endResult = _mm512_kor(endResult, tempRes)
    if endRes == 65535 then
        break
return endResult
```

A 1.2 *4 vs 4*

Algorithm 8: _m512 intersect(DOP a1,..,a4, DOP b1,..,b4)

```
_mm512 endResult
/ for i=0; i<k/2; i++ do
    _mm512 oriAL = _mm512_set_ps(a1[k/2+i],...,a4[k/2+i])
    _mm512 oriBL = _mm512_set_ps(b1[i],...,b4[i])
    _mm512 resL = _mm512_cmp_ps(oriAL, oriBL, _CMP_LT_OS)
    _mm512 oriAH = _mm512_set_ps(a1[i],...,a4[i])
    _mm512 oriBH = _mm512_set_ps(b1[k/2+i],...,b4[k/2+i])
    _mm512 resH = _mm512_cmp_ps(oriAH, oriBH, _CMP_GT_OS)
    _mm512 tempRes = _mm512_kor(resL,resH)
    endResult = _mm512_kor(endResult, tempRes)
    if endResult == 65535 then
        break
return endResult
```

A 2 16-bit to 32-bit floating point conversion

```
#include <immintrin.h>

float f16Tof32(unsigned short n) {
return _cvtsh_ss(n);
}

__m128 f16Tof32(unsigned short) {
__m128i zero = _mm_setzero_si128();
zero = _mm_insert_epi16(zero, edi, 0);
return _mm_cvtph_ps(zero);
}
```

The 14th Gesellschaft für Informatik e.V. (GI) workshop on the Virtual Reality and Augmented Reality (VR/AR)
Tuebingen, Germany, November 17-18, 2017

SIMD Optimized Bounding Volume Hierarchies for Collision Detection

Toni Toni, René Weller, Gabriel Zachmann

University of Bremen

Abstract: We present a novel data structure for SIMD optimized simultaneous bounding volume hierarchy (BVH) traversals like they appear for instance in collision detection tasks. In contrast to all previous approaches, we consider both the *traversal* algorithm and the *construction* of the BVH. The main idea is to increase the branching factor of the BVH according to the available SIMD registers. This requires a novel BVH construction method because traditional BVHs for collision detection usually are simple binary trees. To do that, we present a new BVH construction method based on a clustering algorithm, Batch Neural Gas, that is able to build efficient n-ary tree structures. Our results show that our new data structure outperforms binary trees significantly.

Keywords: Collision Detection, Bounding Volume Hierarchy, SIMD, Advanced Vector Extensions, Batch Neural Gas

1 Introduction

Bounding Volume Hierarchies (BVHs) are widely used to accelerate geometric intersection queries like ray tracing, visibility computations or collision detection. The basic idea is simple: instead of calculating slow and complex geometric intersection tests between all geometric primitives, we wrap them recursively into simple *bounding volumes(BVs)* such as spheres, *axis-aligned bounding boxes (AABB)*, *oriented bounding boxes (OBB)* or *discrete oriented polytopes (k-DOP)*, that allow faster intersection tests. This generates a tree data structure with a single large BV at the root position that encloses all geometric primitives. Obviously, the geometric primitives are the leaves of such a BVH.

The traversal depends on the application. For ray tracing, all rays are tested for intersection with the BVH. Collision detection is more complex: Here, we usually have *two* BVHs that we want to check for intersection, one for each object. We start with the root nodes and simultaneously traverse recursively the children in case of intersection of the BVs (see Algorithm 1 and Figure 1).

Following the trend of acceleration by parallelization it is obvious to apply this idea also to BVH traversals. For ray tracing, this is almost trivial: instead of testing each ray individually, we can simply test them all in parallel. Hence, we only have to change the *traversal* function to perfectly adapt for parallelization. However, parallelization of the simultaneous traversal for collision detection is not obvious. Actually, due to their recursive nature, BVHs are not very well suited for massively parallel acceleration on the GPU. Especially in collision

Algorithm 1: BVHtraversal(BV a, BV b)

```
if a and b are both leaves then
    checkPrimitives(a, b)
else if a is leaf then
    forall children b_i of b do
        if a and b_i intersect then
            BVHtraversal(a, b_i)
else if b is leaf then
    forall children a_i of a do
        if a_i and b intersect then
            BVHtraversal(a_i, b)
else
    forall children a_i of a and b_i of b do
        if a_i and b_i intersect then
            BVHtraversal(a_i, b_i)
```

detection for rigid bodies, where the BVHs do not change by deformations and thus, do not need to be updated or re-constructed, CPU algorithms are still faster than GPU-based methods. However, this simultaneous traversal can still benefit from the SIMD instruction sets of modern CPUs. We can

1. simply switch on a compiler option and hope that the compiler will do the optimization,
2. optimize the *traversal* function manually, depending on the chosen BVH,
3. or adapt the complete BVH structure which additionally requires a redesign of the BVH *construction*.

In this paper we have implemented and tested all of these three methods. There is only one suitable function in Algorithm 1 to optimize the traversal without changing the tree structure: the test for intersection of two BVs. Hence, the benefit of SIMD optimization relies on the type of the BV. For two spheres, we simply have to compute the distance of two points and compare it to the sum of the spheres' radii. This is not very well suited for the for SIMD parallelization because of the length of current AVX2 registers that are able to store eight floating point values. As a consequence, the intersection test for two spheres can be hardly optimized for SIMD. Similarly, the intersection test for AABBs requires four comparisons. Modern AVX registers compare eight float value in a single instruction and this number will increase with upcoming CPU generations. Hence, these BVs could benefit only from the third method, an optimized BVH, but hardly from a simple optimization of the traversal. Consequently, we decided to use a BV that naturally supports all three methods: the k-DOP. Basically, k-DOPs are an extension of AABBs to arbitrary orientations [Zac98].

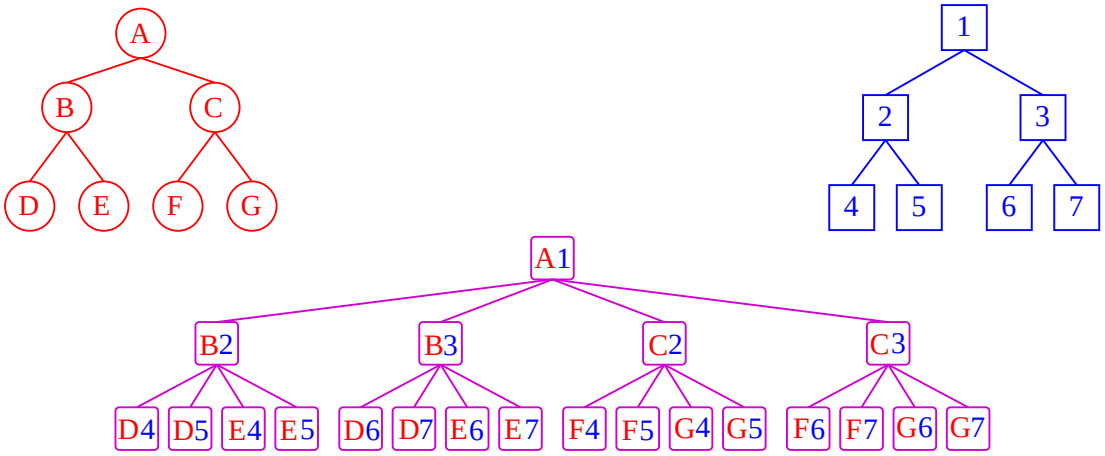


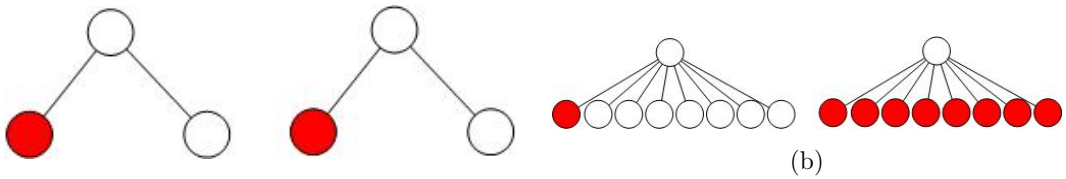
Figure 1: The simultaneous recursive traversal of two binary BVHs during the collision check results in a bounding volume test tree.

They offer a natural trade-off between tightness of the BV and computation time for the intersection test. They show comparable performance to other kinds of bounding volumes [TWZ07]. By choosing the number of orientations k according to the SIMD instruction set, it is straight forward to adapt this BV-type to further SIMD developments.

However, this simple SIMD-parallelization still tests only two BVs in one instruction (see Figure 2a). Hence, it can be applied to almost all existing k-DOP-based BVHs that typically use a binary tree. However, we can also parallelize it in a way that one BV of the first BVH is tested simultaneously against *all* children of the other BVH (see Figure 2a). This is exactly the idea of our new data structure that we call *SIMDop*. In order to take full advantage of SIMD in this case we additionally have to change the branching factor of the tree. This is non trivial because traditional BVH construction methods, like SAH, median-, or mid point-split, that assign the primitives into the sub-trees are not suitable for higher branching factors. Consequently, we have developed new BVH construction methods, this includes simple heuristics but also a new method that is based on Batch Neural Gas clustering. The advantage of such n-ary trees it not only the SIMD accelerated traversal. Additionally, we get less children than with binary trees and the children are also smaller. We have implemented our novel SIMDop BVH and the results show that it outperforms traditional binary trees significantly.

2 Previous Work

In many fields of computer science, BVHs has been used widely to accelerate intersection computation. Usual BVs for the BVHs are spheres [Hub96], AABBs [vdB98] and their memory optimized derivative called BoxTree [Zac02] that is closely related to kd-Trees, k-DOPs [KHM+98, Zac98], a generalization of AABBs, OBBs [GLM96] or convex hull trees [EL01]. Additionally, a wide variety of special BVs for special applications has been developed. For



(a) (b)

Figure 2: The strategy we used for collision query algorithm (a) classic collision query tests only one node at one time. (b) our approach tests one node against eight nodes at one time.

instance spherical shells [KPLM98], swept spheres [LGLM99], spheres that are cut by two parallel planes called slab cut balls [LAM09], quantised orientation slabs with primary orientations (QuOSPO) trees [He99] that combine OBBs with k-DOPs, or combinations of spherical shells with OBBs that was proposed by [KGL+98] for objects that are modelled by Bezier patches.

Usually, a BVH is constructed in a pre-processing step that can be computationally more or less expensive. Basically, there exist two major strategies to build BVHs: bottom-up and top-down. The bottom-up approach starts with elementary BVs of leaf nodes and merges them recursively together until the root BV is reached. A very simple merging heuristic is to visit all nearest neighbours and minimize the size of the combined parent nodes in the same level [RL85]. Less greedy strategies combine BVs by using tilings [LEL97].

However, the most popular method is the top-down approach. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. The main problem is to choose a good splitting criterion. A classical splitting criterion is to simply pick the longest axis and split it in the middle of this axis. Another simple heuristic is to split along the median of the elementary bounding boxes along the longest axis. However, it is easy to construct worst case scenarios for these simple heuristics. The *surface area heuristic (SAH)* tries to avoid these worst cases by optimizing the surface area and the number of geometric primitives over all possible split plane candidates [GS87]. Originally developed for ray tracing, it is today also used for collision detection. The computational costs can be reduced to $O(n \log n)$ [WH06, Wal07] and there exists parallel algorithms for the fast construction on the GPU [LGS+09]. Many other splitting criteria were compared by [Zac00].

The influence of the trees' branching factor is widely neglected in the literature. Usually, most authors simply use binary trees for collision detection. According to [ZL03], the optimum branching factor can be larger. [MKE03] stated that, especially for deformable objects, 4-ary trees or 8-ary could improve the performance. This is mainly due to fewer BV updates. To our knowledge, there does not exist any work that investigates the influence of the branching factor of the BVH for simultaneous traversal tasks.

3 Our SIMDop Data Structure

The main idea of our SIMDop data structure is to construct BVHs with higher branching factor that can be later used during run-time in a SIMD optimized traversal algorithm. Hence, the core is the *construction* that is typically done in a pre-processing step. We propose different methods to construct such n-ary BVHs.

3.1 BVH Construction

We decided to use a top-down approach for the hierarchy construction. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. Moreover, we use a *wrapped hierarchy* according to the notion of [AGN+04], where inner nodes are tight BVs for all their leaves, but they do not necessarily bound their direct children. Compared to layered hierarchies, the big advantage is that the inner BVs are tighter. The main challenge is to choose a good splitting criterion especially, because traditional splitting criteria like SAH do not work for n-ary trees. We propose several splitting criteria for higher branching factors that we will shortly sketch in the following sections.

3.1.1 Longest Axis Split

A classical splitting criterion for binary trees is to sort the primitives along all coord axis and simply pick the longest axis and split this sorted list in the middle of this axis. Obviously, we can easily extend this two n-ary trees by not splitting in the middle, but split the number of BVs into n equal parts. However, this leads to fairly well balanced trees (see Figure 3).

3.1.2 3-Level Longest Axis Split

This is an extension to the longest axis split for n-ary trees where n is preferably in the power of two. We do not simply split along one axis but perform in the first stage a binary longest axis split and than recursively split the primitive sets again until we reach n. In other words, we perform a traditional binary tree split but remove the not needed nodes: instead, we can directly mount all children to the parent node.

3.1.3 Middle Split

This splitting strategy is best suitable for 8-ary trees. The basic idea is related to octrees: We divide the bounding box in the middle of each axis and assign the primitives accordingly.

3.1.4 Median Split

This splitting strategy is also similiar to middle split. However, instead of dividing the bounding box in the middle of each axis, we sort the polygons along each axis. The we compute the median for each axis and set the position for the splitting axis with respect to

the position of the median polygons. Like the middle split, this strategy is best suited for 8-ary trees.

3.1.5 Batch Neural Gas Clustering

Clustering algorithms, especially BNG, have shown to be very efficient for BVH constructions of 4-ary trees [WMS+14]. A nice property of BNG is that it exhibits very robust behavior with respect to the initial cluster center position in contrast to other clustering algorithms like k-means. However, in the original work, the authors used spheres as basic primitives instead of more usual polygonal representations. We simply used the centers of the polygons instead of the spheres' centers reported in the original work in our polygonal implementation. We did not use magnification control, which additionally considers the size of the spheres to produce better clustering results. However, this can be easily added in the future to our polygon-based BNG.

Figure 3 shows the first splitting level for all our splitting criteria.

3.2 BVH Traversal

The idea of the traversal is to test one BV of object A against n BVs of object B. Before the check, we use the method proposed by [Zac98] to realign one of the DOPs in case of affine transformations. A naive implementation for the actual DOP comparison using the current AVX2 instruction set looks as follows, assuming that we are using DOPs with k orientations for the BVs:

Algorithm 2: _m256 intersect(DOP a, DOP b1,...,b8)

```
_mm256 endResult
for i=0; i<k/2; i++ do
    _mm256 oriAL = _mm256_set1_ps(a[i])
    _mm256 oriBL = _mm256_set_ps(b1[i],...,b8[i])
    _mm256 resL = _mm256_cmp_ps(oriAL, oriBL, _CMP_LT_OS)
    _mm256 oriAH = _mm256_set1_ps(a[k/2+i])
    _mm256 oriBH = _mm256_set_ps(b1[k/2+i],...,b8[k/2+i])
    _mm256 resH = _mm256_cmp_ps(oriAH, oriBH, _CMP_GT_OS)
    _mm256 tempRes = _mm256_or_ps(resL,resH)
    endResult = _mm256_or_ps(endResult, tempRes)
    if _m256_movemask_ps(endRes) == 255 then
        break
return endResult
```

There are some drawbacks of this SIMD implementation: First, we have to copy all the values of the DOPs to AVX registers. Second, we have to combine the results using *or*-

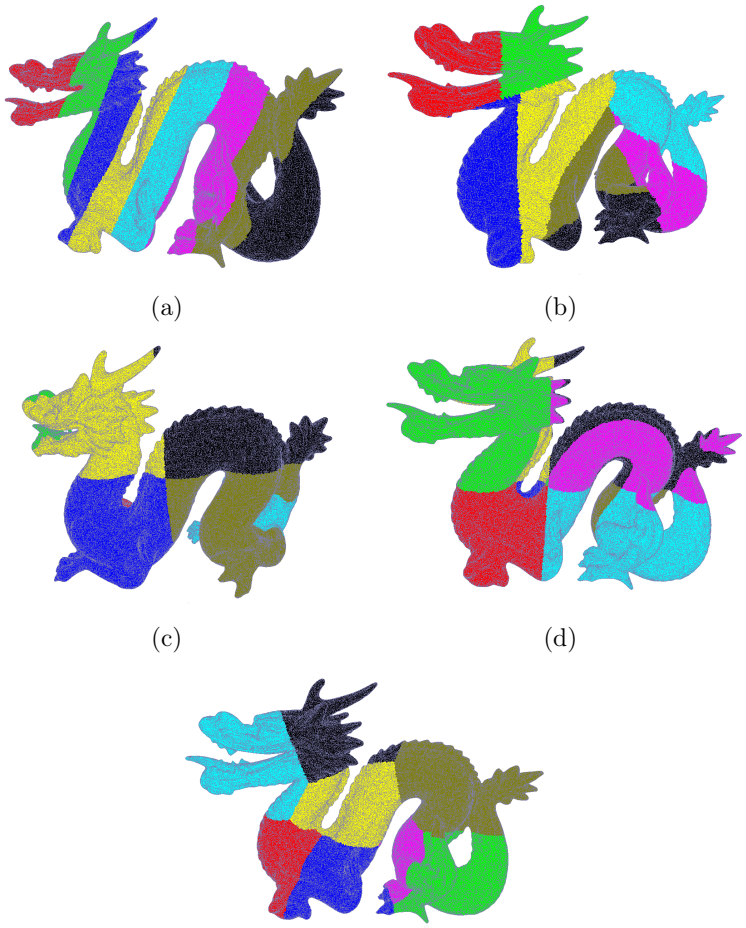
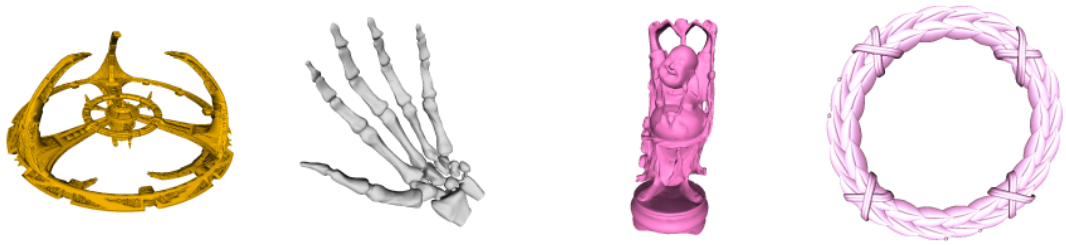


Figure 3: The results of our hierarchy construction algorithms: (a) longest axis, (b) 3 level longest axis, (c) middle split (d) median split (e) Batch Neural Gas.

instructions to compile temporal results the end result. A non-parallel version to check two DOPs for overlap would simply compare two values and use one boolean operation. Hence, in this naive implementation we would need 9 AVX instructions vs. 3 instructions in the non-AVX implementation to compare one orientation of the DOP. Moreover, the non-AVX version could escape the loop earlier for some of the 8 children whereas we have to iterate the loop $k/2$-times if only one of the 8 children overlaps the other DOP. Hence, we could assume an acceleration of at most $\frac{3\times 8}{9}$, because we test 8 children simultaneously, not considering the faster loop escapes but also the smaller BVs of the SIMDop structure.

3.3 Optimization

Our benchmarks have shown, that actually, our naive implementation performs worse than the non-AVX version. The main reason is that the *_mm256_load_ps* and *_mm256_load_ps1* instructions that loads the data into the AVX registers requires more time than the other in-



(a) (b) (c) (d)

Figure 4: The objects we used in our timings: (a) ds9 space station, (b) hand, (c) happy buddha, and (d) laurel.

structions. However, we can easily solve this by directly storing the values into a proper AVX format. This would lead to a theoretical benefit of $\frac{3\times8}{5}$ because we only need 5 instructions per orientation. Unfortunately, this increases the memory footprint: Theoretically, without optimization, each node holds values of it's eight children. Here we need one floating point value for each orientation of each child which results in $8 \times k \times 4$ Bytes for 32-bit floating point values in case of 8 children. In order to avoid the *_mm256_load_ps1* instruction, we have to copy each value of each child 8 times. Overall, this would increase the total memory footprint by a factor of 9.

In order to overcome this drawback somewhat we propose a second optimization: Actually it is sufficient to provide conservative overlap tests, i.e., a few more false positives for the BVH traversal do not hurt so much. Hence, we can lower the resolution of the DOPs by using half precision floats (16-bit floating-point numbers) [Kon12], which are half the size of traditional 32-bit single precision floats. Thus we could save half the storage space and half the memory bandwidth of the bounding volume.

It seems to be obvious to directly use the reduced accuracy to increase the branching factor even more to a factor of 16. Unfortunately, current SIMD instructions sets do not support comparisons of half precision float values directly. Hence, before the actual intersection test, we have to convert the 16-bit half floats back to 32-bit float values. Since the 3rd generation of Intel® Core™ processors, this conversion is supported by the *vcvtps2ph* instruction.

4 Results

We have implemented our algorithms using C++ and *Intel Intrinsics function* using Visual Studio 2015. We focused our implementation on the most recent AVX2 instruction sets. For the binary tree we used the AVX2 compiler flag. All tests were performed on a system with an Intel I7 6700 CPU, 8GB of main memory and a NVIDIA Geforce GTX 660 GPU with 2GB of memory. We used the benchmarking suite proposed by [TWZ07]. Figure 4 shows some of the used models with different shapes and resolutions in our timings: in particular, a ds9 space station, a hand, a happy Buddha and a laurel. For the hand optimized traversal

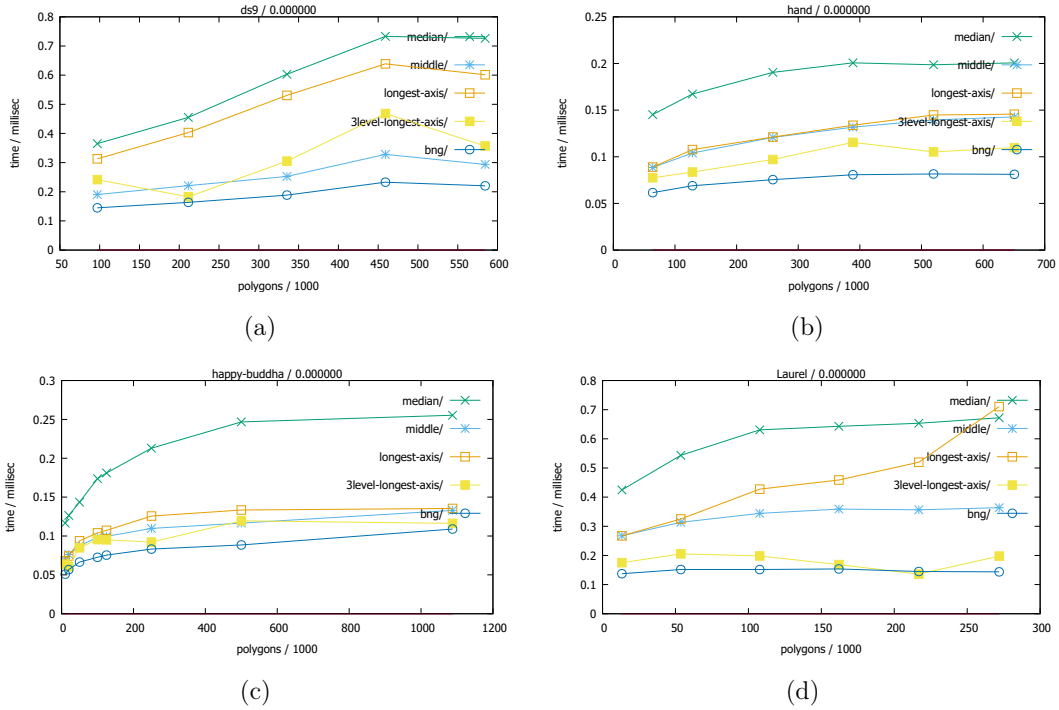


Figure 5: Average timing of collision queries the different splitting criteria of our SIMDop BVH for the object (a) ds9 space station, (b) hand, (c) happy Buddha, and (d) laurel. The BVH constructed with BNG is up to four times faster than the slowest BVH.

method with the binary tree we used a k of 32 orientations and for the 8-ary DOP 46 orientations. Obviously, for the binary tree, k should be divisible by 8, however, for the 8-ary tree this is not mandatory. In all our timings, these k values performed best in all our test cases for the particular trees. In all our timings presented in the plots presented here we used a distance of zero because this is the most time consuming configuration distance according to [TWZ07].

First, we investigated the influence of the splitting criterion described in Section 3.1. The BNG clustering outperforms the other heuristics significantly, independent of the objects' shapes (see Figure 6).

Second, we investigated the influence of using half floats instead of full 32-bit floating point values. We were able to gain additional 5 - 10 % performance speed-up (See Figure 7) while reducing the memory usage by 50% at the same time.

Finally, we compared the performance of our SIMDop BVHs to the other methods, i.e. the binary tree-based data structures with the the compiler flag SIMD optimization and the manually SIMD-optimized traversal algorithm. The compiler flag optimized binary DOP tree and the manually AVX optimized DOP tree traversal have very similar running times. This gives a hint that compiler optimization seems to work very well. However, our SIMD optimized SIMDop data structure outperforms both binary DOP trees by at least factor of

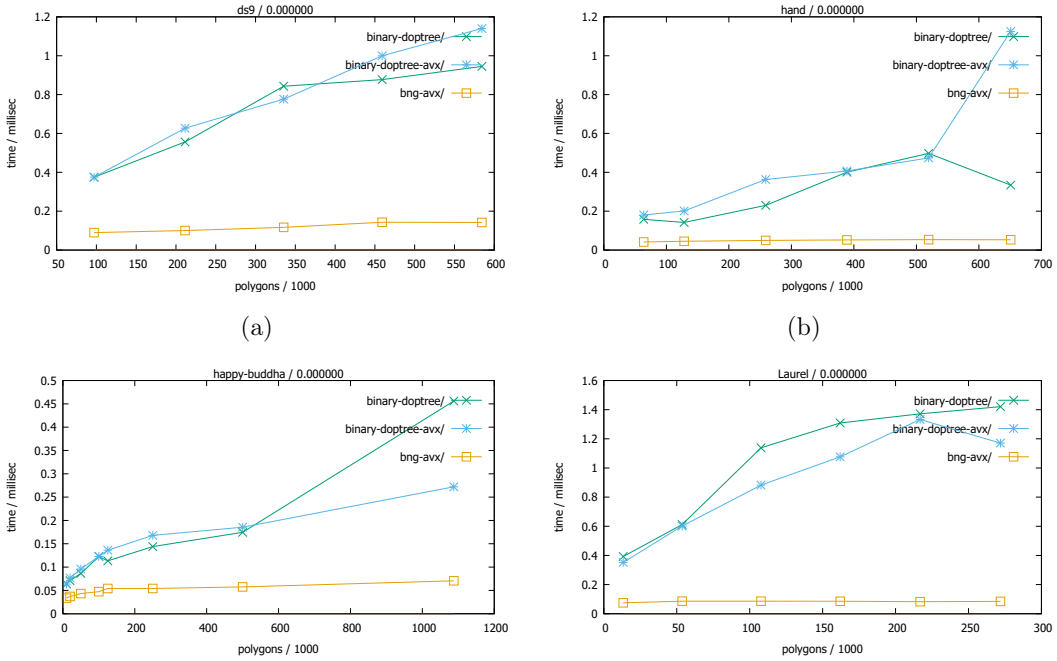


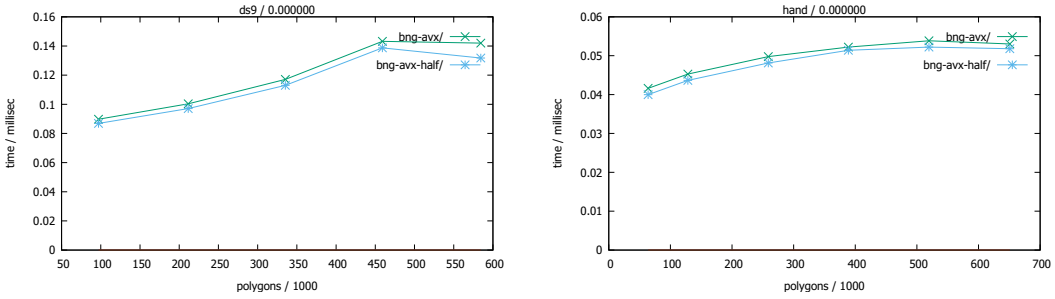
Figure 6: Average timing of collision queries for AVX implementation of our SIMDop BVH (bng-avx) for all our test objects compared with manually optimized binary DOP tree (binary-doptree-avx) and the compiler optimized binary DOP tree. The results show that our SIMDop BVH is up to ten times faster than both binary DOP trees.

10 for the ds9 station, a factor of 7 for the hand, a factor of 6 for the Buddha statue and a factor of 10 for the laurel. In all cases this factor increases with an increasing polygon count.

5 Conclusions and Future Work

We have presented a novel SIMD optimized bounding volume hierarchy for simultaneous BVH traversal. The main idea is to use higher-ary trees instead of classical binary tress. We have presented several new heuristics for the top-down construction of such tree data structures with higher branching factor. The BNG-based method performs best. Even if we tested only 8-ary trees, the clustering-based construction is already prepared to support higher branching factors following future SIMD developments. Our results show that, depending on the object, our SIMDop BVH outperforms traditional BVHs by more than an order of magnitude.

Our approach also opens up several directions for future work. For instance, it would be interesting to include magnification control to the BNG construction algorithm. Moreover, other clustering algorithms than BNG could be considered. In this work, we only considered DOPs as BVs because of a fair comparison with the manual optimized traversal scheme. However, investigating other BV types could also be interesting. Finally, probably also



(a) (b)

Figure 7: Average timing for collision queries using half float vs 32-float for our SIMDop data structure for the ds station and the hand object. We get a performance gain of 5-10% while dividing the memory footprint by half.

other applications using BVHs like ray tracing or occlusion computations could benefit from our SIMDop BVH.

References

[AGN+04] Pankaj Agarwal, Leonidas Guibas, An Nguyen, Daniel Russel, and Li Zhang. Collision detection for deforming necklaces. *Computational Geometry: Theory and Applications*, 28:137–163, 2004.

[EL01] Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of EUROGRAPHICS 2001)*, 20(3):500–510, 2001.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 171–180, New York, NY, USA, 1996. ACM.

[GS87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, 7(5):14–20, May 1987.

[He99] Taosong He. Fast collision detection using quospo trees. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, I3D '99, pages 55–62, New York, NY, USA, 1999. ACM.

[Hub96] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph.*, 15(3):179–210, 1996.

[KGL+98] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using shelltrees, 1998.

[KHM+98] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, January 1998.

[Kon12] Patrick Konsor. Performance benefits of half precision floats, 2012.

[KPLM98] Shankar Krishnan, Amol Pattekar, Ming C. Lin, and Dinesh Manocha. Spherical shell: a higher order bounding volume for fast proximity queries. In *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 177–190, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[LAM09] Thomas Larsson and Tomas Akenine-Möller. Bounding volume hierarchies of slab cut balls. *Comput. Graph. Forum*, 28(8):2379–2395, 2009.

[LEL97] Scott T. Leutenegger, Jeffrey M. Edgington, and Mario A. Lopez. Str: A simple and efficient

algorithm for r-tree packing. Technical report, Institute for Computer Applications in Science and Engineering (ICASE), 1997.

[LGLM99] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes, November 14 1999.

[LGS+09] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David P. Luebke, and Dinesh Manocha. Fast bvh construction on gpus. *Computer Graphics Forum*, 28(2):375–384, 2009.

[MKE03] Johannes Mezger, Stefan Kimmerle, and Olaf Etzmuß. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2):322–329, 2003.

[RL85] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed r-trees. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, SIGMOD '85, pages 17–31, New York, NY, USA, 1985. ACM.

[TWZ07] Sven Trenkel, René Weller, and Gabriel Zachmann. A benchmarking suite for static collision detection algorithms. In Václav Skala, editor, *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, Plzen, Czech Republic, 29 January–1 February 2007. Union Agency.

[vdB98] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, January 1998.

[Wal07] Ingo Wald. On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 33–40, Washington, DC, USA, 2007. IEEE Computer Society.

[WH06] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). *Symposium on Interactive Ray Tracing*, 0:61–69, 2006.

[WMS+14] René Weller, David Mainzer, Abhishek Srinivas, Matthias Teschner, and Gabriel Zachmann. Massively parallel batch neural gas for bounding volume hierarchy construction. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)*, Bremen, Germany, September 2014. Eurographics Association.

[Zac98] Gabriel Zachmann. Rapid collision detection by dynamically aligned dop-trees. In *Proceedings of the Virtual Reality Annual International Symposium*, VRAIS '98, pages 90–, Washington, DC, USA, 1998. IEEE Computer Society.

[Zac00] Gabriel Zachmann. *Virtual Reality in Assembly Simulation – Collision Detection, Simulation Algorithms, and Interaction Techniques.* Dissertation, Darmstadt University of Technology, Germany, May 2000.

[Zac02] Gabriel Zachmann. Minimal hierarchical collision detection. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '02, pages 121–128, New York, NY, USA, 2002. ACM.

[ZL03] Gabriel Zachmann and Elmar Langetepe. Geometric data structures for computer graphics. In *Proc. of ACM SIGGRAPH*. ACM Transactions of Graphics, 27–31July 2003.

2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Macau, China, November 4-8, 2019

SIMDop: SIMD Optimized Bounding Volume Hierarchies for Collision Detection

Toni Tan[1], René Weller[1] and Gabriel Zachmann[1]

***Abstract*— We present a novel data structure for SIMD optimized simultaneous bounding volume hierarchy (BVH) traversals like they appear for instance in collision detection tasks. In contrast to all previous approaches, we consider both the *traversal* algorithm and the *construction* of the BVH. The main idea is to increase the branching factor of the BVH according to the available SIMD registers and parallelize the simultaneous BVH traversal using SIMD operations. This requires a novel BVH construction method because traditional BVHs for collision detection usually are simple binary trees. To do that, we present a new BVH construction method based on a clustering algorithm, Batch Neural Gas, that is able to build efficient n-ary tree structures along with SIMD optimized simultaneous BVH traversal. Our results show that our new data structure outperforms binary trees significantly.**

I. Introduction

Collision detection (CD) algorithms are essential for sampling-based motion planning algorithms. They are used to test whether a sampled configuration is in collision with the workspace obstacles. In most sampling-based motion planning algorithms, the collision computation is the computational bottleneck that requires up to 90% of computation time [1].

For most CD algorithms that work with polyhedral models, *Bounding Volume Hierarchies (BVHs)* are the common technique used to accelerate the intersection queries. The basic idea is simple: instead of calculating slow and complex geometric intersection tests between all geometric primitives, we wrap them recursively into simple *bounding volumes (BVs)* such as spheres, *axis-aligned bounding boxes (AABB)*, *oriented bounding boxes (OBB)* or *discrete oriented polytopes (k-DOP)*, that allow faster intersection tests. This generates a tree data structure with a single large BV at the root position that encloses all geometric primitives. Obviously, the geometric primitives are the leaves of such a BVH.

As for the traversal, we usually have *two* BVHs that we want to check for intersection, one for each object. We start with the root nodes and simultaneously traverse recursively the children in case of intersection of the BVs (see Algorithm 1 and Figure 1).

Following the trend of acceleration by parallelization it is obvious to apply this idea also to BVH traversals. Unfortunately, the parallelization of the simultaneous traversal for collision detection is not obvious. Actually, due to their recursive nature, BVHs are not very well suited for massively parallel acceleration on the GPU. Moreover, especially for online planning, robots are often not equipped with powerful GPUs.

[1]University of Bremen, Germany

Algorithm 1: BVHtraversal(BV a, BV b)

```
if a and b are both leaves then
    checkPrimitives(a, b)
else if a is leaf then
    forall children b_i of b do
        if a and b_i intersect then
            BVHtraversal(a, b_i)
else if b is leaf then
    forall children a_i of a do
        if a_i and b intersect then
            BVHtraversal(a_i, b)
else
    forall children a_i of a and b_i of b do
        if a_i and b_i intersect then
            BVHtraversal(a_i, b_i)
```

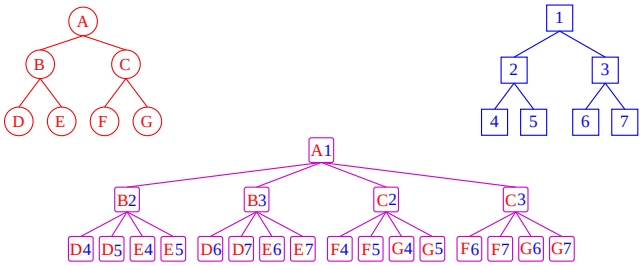


Fig. 1: The simultaneous recursive traversal of two binary BVHs during the collision check results in a bounding volume test tree.

However, the simultaneous traversal required in BVH-based collision detection can still benefit from the SIMD instruction sets of modern CPUs.

To take advantage from this parallel operations, we can:

1) simply switch on a compiler option and hope that the compiler will do the optimization,
2) optimize the *traversal* function manually, depending on the chosen BVH,
3) or adapt the complete BVH structure which additionally requires a redesign of the BVH *construction*.

In this paper we have implemented and tested all of these three methods. There is only one suitable function in Algorithm 1 to optimize the traversal without changing the tree structure: the test for intersection of a pair of BVs. Hence, the benefit of SIMD optimization relies heavily on the

type of the BV. For two spheres, we simply have to compute the distance of two points and compare it to the sum of the spheres' radii. This is not very well suited for the for SIMD parallelization because of the length of current AVX512 registers that are able to store 16 floating point values. As a consequence, the intersection test for two spheres can be hardly optimized for SIMD. Similarly, the intersection test for AABBs requires four comparisons. Modern AVX registers compare 16 float value in a single instruction and this number will increase with upcoming CPU generations. Hence, these BVs could benefit only from the third method, an optimized BVH, but hardly from a simple optimization of the traversal. Consequently, we decided to use a BV that naturally supports all three methods: the k-DOP. Basically, k-DOPs are an extension of AABBs to arbitrary orientations [2]. They offer a natural trade-off between tightness of the BV and computation time for the intersection test. They show comparable performance to other kinds of bounding volumes [3]. By choosing the number of orientations k according to the SIMD instruction set, it is straightforward to adapt this BV-type to further SIMD developments.

However, this simple SIMD-parallelization still tests only two BVs in one instruction (see Figure 2a). Hence, it can be applied to almost all existing k-DOP-based BVHs that typically rely on a binary tree. However, we can also parallelize it in a way that one BV of the first BVH is tested simultaneously against *all* children of the other BVH (see Figure 2a). This is exactly the idea of our new data structure that we call *SIMDop*. In order to take full advantage of SIMD in this case we additionally have to change the branching factor of the tree. This is non trivial because traditional BVH construction methods, like the *surface area heuristic (SAH)*, median-, or mid point-split, that assign the primitives into the sub-trees are not suitable for higher branching factors. Consequently, we have developed new BVH construction methods, this includes simple heuristics but also a new method that is based on Batch Neural Gas clustering. The advantage of such n-ary trees is not only the SIMD accelerated traversal. Additionally, we get less children than with binary trees and the children are also smaller. We have implemented our novel SIMDop BVH and the results show that it outperforms traditional binary trees by an order of magnitude.

II. Previous Work

In many fields of computer science, BVHs has been used widely to accelerate intersection computation. Usual BVs for the BVHs are spheres [4], AABBs [5] and their memory optimized derivative called BoxTree [6] that is closely related to kd-Trees, k-DOPs [7], [2], a generalization of AABBs, OBBs [8] or convex hull trees [9]. Additionally, a wide variety of special BVs for special applications has been developed. For instance spherical shells [10], swept spheres [11], spheres that are cut by two parallel planes called slab cut balls [12], quantised orientation slabs with primary orientations (QuOSPO) trees [13] that combine OBBs with k-DOPs, or combinations of spherical shells with OBBs that

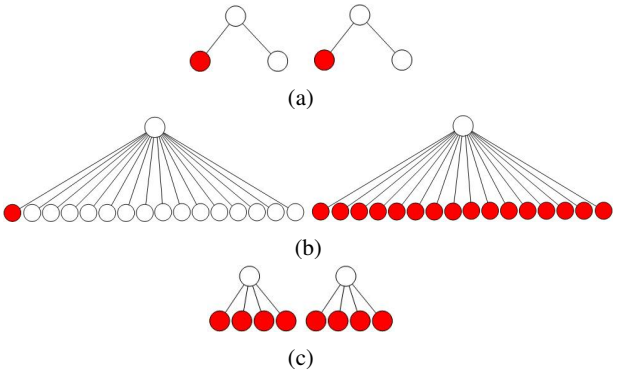


Fig. 2: Different strategy to compute intersections of the child nodes in the simultaneous traversal algorithm: (a) classic collision query tests only one pair of nodes. With SIMD, assuming 16 registers, we can (b) test one node of the left BVH against 16 nodes of the right BVH simultaneously in the case of a branching factor of 16 or (c), in case of a branching factor of 4, test all nodes from same level at one time.

was proposed by [14] for objects that are modelled by Bezier patches.

In sampling-based motion planning, AABB-based BVHs are widely used to calculate collision between candidate trajectories in workspace [15]. The *Flexible Collision Library (FCL)* [16] also supports several BVs for its BVH such as AABB, OBB, rectangle swept spheres (RSS) and k-DOPs. Another approach using a hierarchical three-stage sequence of BVs namely AABB, Sphere, and OBBs [17].

Usually, a BVH is constructed in a pre-processing step that can be computationally more or less expensive. Basically, there exist two major strategies to build BVHs: bottom-up and top-down. The bottom-up approach starts with elementary BVs of leaf nodes and merges them recursively together until the root BV is reached. A very simple merging heuristic is to visit all nearest neighbours and minimize the size of the combined parent nodes in the same level [18]. Less greedy strategies combine BVs by using tilings [19].

However, the most popular method is the top-down approach. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. The main problem is to choose a good splitting criterion. A classical splitting criterion is to simply pick the longest axis and split it in the middle of this axis. Another simple heuristic is to split along the median of the elementary bounding boxes along the longest axis. However, it is easy to construct worst case scenarios for these simple heuristics. SAH tries to avoid these worst cases by optimizing the surface area and the number of geometric primitives over all possible split plane candidates [20]. Originally developed for ray tracing, it is today also used for collision detection. The computational costs can be reduced to $O(n \log n)$ [21], [22] and there exists parallel algorithms for the fast construction on the GPU [23]. Many other splitting criteria were compared in [24].

The influence of the trees' branching factor is widely neglected in the literature. Usually, most authors simply use binary trees for collision detection. According to Zachmann and Langetepe [25], the optimum branching factor can be larger. Mezger et al. [26] stated that, especially for deformable objects, 4-ary trees or 8-ary could improve the performance. This is mainly due to fewer BV updates. To our knowledge, there does not exist any work that investigates the influence of the branching factor of the BVH for simultaneous traversal tasks.

III. SIMD Recap

Originally, *SIMD instruction sets* had been introduced to support integer computation for intensive multimedia applications, but later they have been extended to support floating point computation which extends the usefulness also for scientific computations. The idea is that a single instruction operates on different input data values (e.g. 8 or 16 floating point values) simultaneously. Several slightly different SIMD instruction sets are available for various CPUs; e.g. NEON for Arm based CPUs and SSE/AVX for both Intel and AMD CPUs (see Table I for a list of available SIMD instruction sets and the supported data types). The most current AVX512 instruction set supports computation of 16 single precision-float in parallel. In this paper, we focus on mainly AVX512, however the idea can be easily implemented on other SIMD instruction sets such as SSE, AVX, and NEON. Moreover, we included measurements for AVX in our results and we are confident, that more powerful AVX registers will be available for the other platforms soon.

Name	Width	Types	supported CPUs
NEON	128 bits	4x single 2x double*	Armv7-A/R and above *only available for Armv8-A
SSE	128 bits	4x single	Intel Pentium III and above AMD Athlon XP and above
SSE2 SSE3 SSE4	128 bits	4x single 2x double	Intel Pentium 4 and above AMD Athlon XP and above
AVX AVX2	256 bits	8x single 4x double	Intel Sandy Bridge and above AMD Bulldozer and above
AVX512	512 bits	16x single 8x double	Intel Skylake-X and above

TABLE I: Floating point support for various SIMD Instruction Sets

IV. Our SIMDop Data Structure

The main idea of our SIMDop data structure is to construct BVHs with higher branching factor that can be later used during run-time in a SIMD optimized traversal algorithm. Hence, the core is the *construction* that is typically done in a pre-processing step. We propose different methods to construct such n-ary BVHs.

A. BVH Construction

We decided to use a top-down approach for the hierarchy construction. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. Moreover, we use a *wrapped hierarchy* according to the notion of Agarwal et al. [27], where inner nodes are tight BVs for all their leaves, but they do not necessarily bound their direct children. Compared to layered hierarchies, the big advantage is that the inner BVs are tighter. The main challenge is to choose a good splitting criterion especially, because traditional splitting criteria like SAH do not work for n-ary trees. We propose several splitting criteria for higher branching factors that we will shortly sketch in the following sections.

1) Longest Axis Split: A classical splitting criterion for binary trees is to sort the primitives along all coord axis and simply pick the longest axis and split this sorted list in the middle of this axis. Obviously, we can easily extend this two n-ary trees by not splitting in the middle, but split the number of BVs into n equal parts. However, this leads to fairly well balanced trees (see Figure 3).

2) Extended Longest Axis Split: This is an extension to the longest axis split for n-ary trees where n is preferably in the power of two. We do not simply split along one axis but perform in the first stage a binary longest axis split and than recursively split the primitive sets again until we reach n. In other words, we perform a traditional binary tree split but remove the not needed nodes: instead, we can directly mount all children to the parent node.

3) Batch Neural Gas Clustering: Clustering algorithms, especially BNG, have shown to be very efficient for BVH constructions of 4-ary trees [28]. A nice property of BNG is that it exhibits very robust behavior with respect to the initial cluster center position in contrast to other clustering algorithms like k-means. However, in the original work, the authors used spheres as basic primitives instead of more usual polygonal representations. We simply used the centers of the polygons instead of the spheres' centers reported in the original work in our polygonal implementation. We did not use magnification control, which additionally considers the size of the spheres to produce better clustering results. However, this can be easily added in the future to our polygon-based BNG.

Figure 3 shows the first hierarchy level for all our splitting criteria and different branching factors.

B. BVH Traversal

The key part to optimize the traversal in Algorithm 1 is the test for intersection of the child bounding volumes. For binary trees, the four possible combinations of child pairs are usually traversed sequentially. SIMD enables us to accelerate this intersection test in several ways:

- We can use a SIMD instructions to replace a single test of a pair of BVs (see Figure 2a). This would leave the for-loop untouched and just replace the intersection method.
- We can also remove the first part of the for-loop and test one BV of the first BVH simultaneously against *all* children of the other BVH (see Figure 2b). For AVX512 this results in a 1 vs 16 check. Accordingly, we call our BVH using this approach 1vs16-SIMDop.

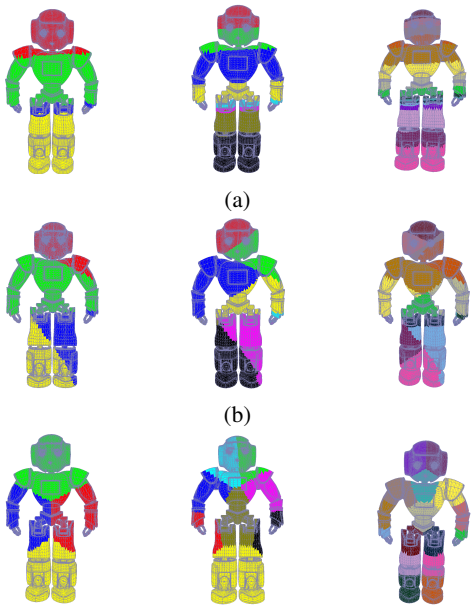


Fig. 3: The results of our hierarchy construction algorithms showing the color coded first level of the hierarchy: (a) longest axis, (b) extended longest axis, and (c) Batch Neural Gas. The trees have degree four on the left side, degree eight in the center, and sixteen on the right side.

- Finally, we can remove all for-loops and test all nodes from the same level at one time (see Figure 2c). With AVX512 this results in a 4 vs 4 test and we call the respective BVH 4vs4-SIMDop.

An implementation of the first idea is straight forward, it requires a simple replacement of the comparison inside the intersection function.

Algorithm 2 shows the naive implementation for the 1vs16-SIMDop, i.e. the removal of the inner for-loop by testing one BV of object A against n BVs of object B, using the current AVX512 instruction set looks as follows, assuming that we are using DOPs with k orientations for the BVs[1].

Removing both loops for the 4vs4-SIMDop, i.e. testing all n child BVs of object A against all n child BVs of object B for a pair of nodes requires just a slightly different ordering of the Dop values which results in the AVX512 code that is shown in Algorithm 3.

There are some drawbacks of these SIMD implementations: first, we have to copy all the values of the DOPs to AVX registers. Second, we have to combine the temporal results using *or*-instructions to compile the end result. A non-parallel version to check two DOPs for overlap would simply

[1]Variables of the type *_mm512* are AVX512 variables with a length of 512 Bit. Intrinsic AVX512 instructions usually start with the prefix *_mm512_* followed by the particular operation and end with a suffix that indicates the data type: e.g. *_mm512_cmp_ps* compares the two input variables of 512 Bit width of the type single precision floating point (*_ps*), following the rule defined in the third parameter and returns the result as a 512 Bit vector. *_mm512_kor* defines a bitwise logical OR comparison using masks.

Algorithm 2: _m512 intersect(DOP a, DOP b1,...,b16)

```
_mm512 endResult
for i=0; i¡k/2; i++ do
    _mm512 oriAL = _mm512_set1_ps(a[i])
    _mm512 oriBL = _mm512_set_ps(b1[i],...,b16[i])
    _mm512 resL = _mm512_cmp_ps(oriAL, oriBL,
      _CMP_LT_OS)
    _mm512 oriAH = _mm512_set1_ps(a[k/2+i])
    _mm512 oriBH = _mm512_set_ps(b1[k/2+i],...,b16[k/2+i])
    _mm512 resH = _mm512_cmp_ps(oriAH, oriBH,
      _CMP_GT_OS)
    _mm512 tempRes = _mm512_kor(resL,resH)
    endResult = _mm512_kor(endResult, tempRes)
    if endRes == 65535 then
        break
return endResult
```

Algorithm 3: _m512 intersect(DOP a1,..,a4, DOP b1,..,b4)

```
_mm512 endResult
/ for i=0; i¡k/2; i++ do
    _mm512 oriAL = _mm512_set_ps(a1[k/2+i],...,a4[k/2+i])
    _mm512 oriBL = _mm512_set_ps(b1[i],...,b4[i])
    _mm512 resL = _mm512_cmp_ps(oriAL, oriBL,
      _CMP_LT_OS)
    _mm512 oriAH = _mm512_set_ps(a1[i],...,a4[i])
    _mm512 oriBH = _mm512_set_ps(b1[k/2+i],...,b4[k/2+i])
    _mm512 resH = _mm512_cmp_ps(oriAH, oriBH,
      _CMP_GT_OS)
    _mm512 tempRes = _mm512_kor(resL,resH)
    endResult = _mm512_kor(endResult, tempRes)
    if endResult == 65535 then
        break
return endResult
```

compare two values and use one boolean operation. Hence, in these naive implementations we would need 9 AVX instructions vs. 3 instructions in the non-AVX implementation to compare one orientation of the DOP. Moreover, the non-AVX version could escape the loop earlier for some of the 16 children whereas in the SIMD cases we have to iterate the loop $k/2$-times if only one of the 16 children overlaps the other DOP. Hence, we could assume an acceleration of at most $\frac{3\times16}{9}$, because we test 16 children simultaneously, not considering the faster loop escapes and the smaller BVs of the SIMDop structure.

C. Optimization

Our benchmarks have shown that actually, our naive implementations for Algorithms 2 and 3 perform worse than the non-AVX version. The main reason is that the *_mm512_set_ps* and *_mm512_set1_ps* instructions that load the data into the AVX registers require more time than the other instructions. For the 1vs16-SIMDop we can easily solve this by directly storing the values into a proper AVX format. This would lead to a theoretical benefit of $\frac{3\times16}{5}$ because we only need 5 instructions per orientation, not considering the smaller BVs.

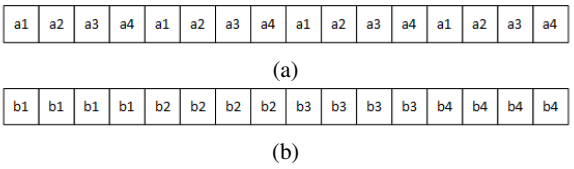


Fig. 4: Permutation of the values for the for DOPs $a1, ..., a4$ of an object A and the four DOPs $b1, ..., b4$ of an object B to produce a single 512 Bit AVX register for comparing all 4×4 possible combinations in Algorithm 3.

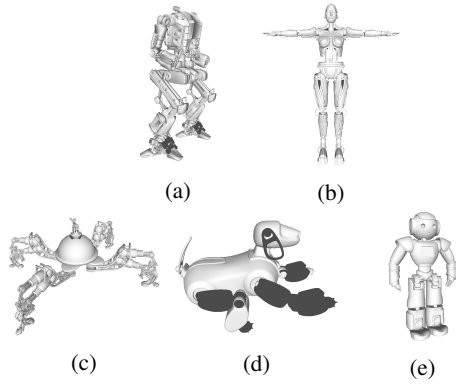


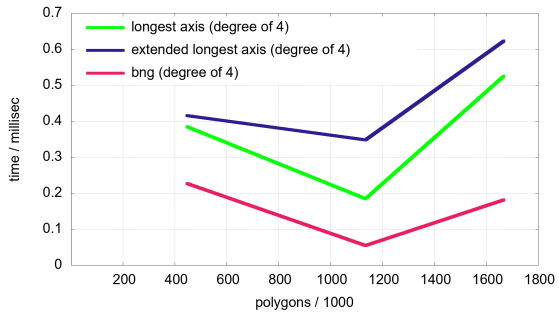
Fig. 5: The objects we used in our timings: (a) ATST walker robot, (b) female robot, (c) quadripod robot, (d) dog robot, and (e) Nao.

Moreover, we tested to use prefetching to improve the cache performance. Unfortunately, the size of a cache line in CPUs supporting AVX512 is exactly 512 Bit, so we did not see any acceleration. However, we were able to reduce the memory bandwidth by storing the DOP values as half floats. Since the 3rd generation of Intel® Core™ processors, the conversion of 16-bit half floats back to 32-bit float values is supported by the *vcvtps2ph* instruction without computational overhead. The resulting increase of false positives for the BVH traversal was neglectable.

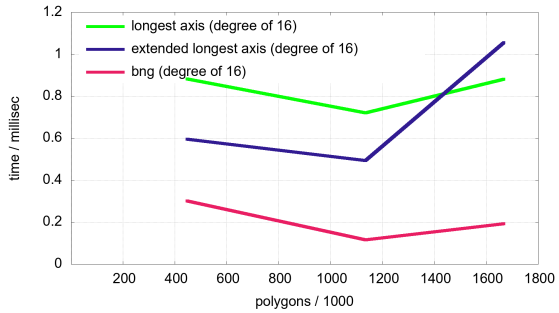
Obviously, we could also store the data for the 4vs4-SIMDop into an appropriate AVX512 variable. However, this would increase the memory footprint by a factor of 4 because we would have to copy each DOP value four times. In order to avoid this waste of memory and to further improve cache performance, we decided to use a different strategy for the 4vs4-SIMDop: AVX512 supports the function *_mm512_castps128_ps512* that casts 128 Bit SSE data to AVX512 data with zero latency. Hence, we store 4 floating point values per DOP orientation and use the permutation function *_mm512_permutexvar_ps* to shuffle the values to their correct positions (see Figure 4). This requires one more AVX instruction, leading to a theoretical benefit of $\frac{3 \times 16}{6}$ compared to the sequential binary tree, but it significantly improves cache performance.

V. Results

We have implemented our algorithms using C++ and *Intel Intrinsics functions* using Visual Studio 2017. We focused our implementation on the most recent AVX512 instruction



(a)



(b)

Fig. 6: Average collision query time for the different splitting criteria for the quadripod robot using (a) the 4vs4-SIMDop and (b) the 1vs16-SIMDop with respect to the polygon count. The results are very similar for all objects.

sets. All tests were performed on a system with an Intel I7 7800X CPU, 64GB of main memory and a NVIDIA Geforce GTX 980 GPU with 4GB of memory. We used the standardized benchmarking suite proposed by Trenkel et al. [3]. Figure 5 shows some of the used models with different shapes and resolutions in our timings: in particular, a ATST walker robot, a female android, a quadripod and a robotic dog. According to Trenkel et al. [3], we present all results in this section for the most time consuming distance preset, i.e. a distance of zero. For the best performance of the hand optimized traversal function of the binary tree, k should be divisible by 16. We set the number of orientations of the DOPs to $k = 32$ where not other mentioned because it performs better than $k = 16$ for all methods.

First, we evaluated the influence of the splitting criterion described in Section IV-A. The BNG clustering outperforms the other heuristics significantly in all our test cases, independent of the objects' shapes and polygon count (see Figure 6). The benefit of the clustering increases with and increasing number of branches in the tree. In term of BVH construction time, the BNG clustering-based SIMDop for both degree of 4 and 16 can be constructed almost as fast BVH constructed using V-COLLIDE and binary DOP tree (see Figure 7).

Moreover, we compared the performance of our two SIMDop variants to the other methods 1 and 2, i.e. the binary tree-based data structures with the compiler flag SIMD optimization and the manually SIMD-optimized traversal algorithm. The compiler flag optimized binary DOP tree

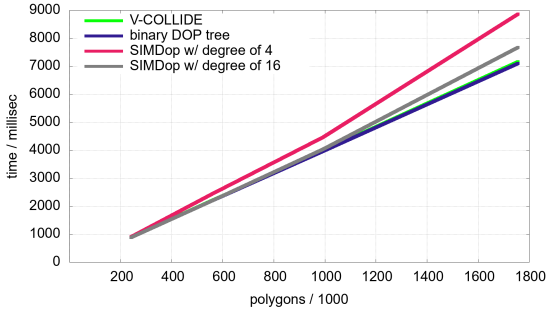
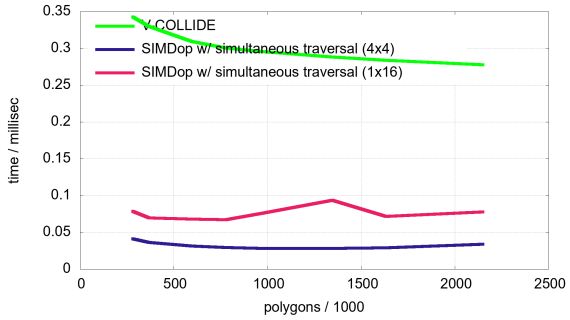
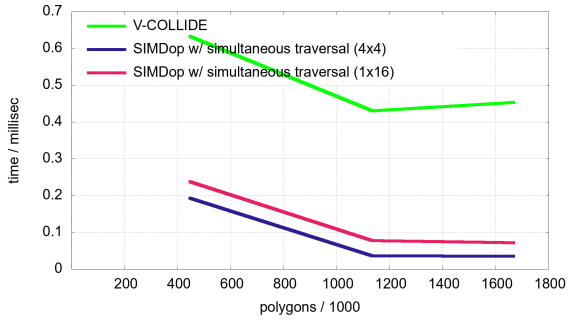


Fig. 7: A comparison of BVH construction time of our SIMDop based on BNG clustering algorithm compared with V-COLLIDE and binary DOP tree for the ATST walker robot.



(a)

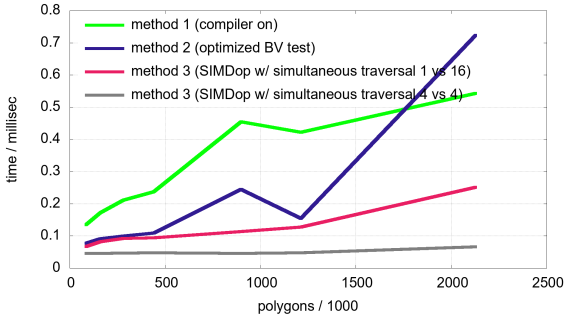


(b)

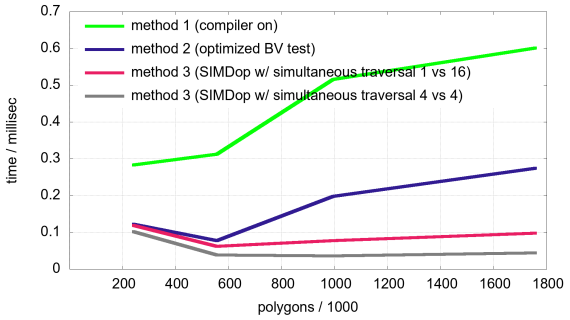
Fig. 8: A comparison of our SIMDop with V-COLLIDE library for (a) female robot, and (b) quadripod. The results show that our 4vs4-SIMDop performs best and faster than V-COLLIDE by up to eight times for (a) female robot and thirteen times for (b) quadripod.

and the manually AVX optimized DOP tree traversal have very similar running times. This gives a hint that compiler optimization seems to work very well. However, our two SIMD optimized SIMDop versions, the 4vs4-SIMDop and 1vs16-SIMDop both outperform both binary DOP trees by at least factor of 8 for the Nao (see Figure 9a), a factor of 13 for the ATST walker robot (see Figure 9b). In all cases this factor increases with an increasing polygon count. This is slightly higher than the theoretical factor of $\frac{3\times16}{5}$ we expected from the number of instructions for the intersection function. This indicates that the decreased size of the BVs due to the higher branching factor and the reduced number of overall BVs in a tree with higher branching factor compensate the increasing number of iterations required for the SIMD loop.

We also compared our SIMDop to the V-COLLIDE library that is often used for sample-based path planning tasks. An experimental comparative analysis has shown that V-COLLIDE outperforms other CD libraries like PQP [29]. Figure 8 shows that our 4vs4-SIMDop is able to outperform V-COLLIDE by a factor of up to 13 for the quadropod.



(a)



(b)

Fig. 9: Average collision query times for the compiler optimized binary DOP tree (method 1), the manually optimized binary DOP tree (method 2) and the two SIMD optimized Dop tree versions, the 1vs16-SIMDop and the 4vs4-SIMDop (method 3) with respect to the number of polygons in the (a) Nao and (b) ATST walker robot. The results show that our SIMDop are up to eight and thirteen times faster than both binary DOP trees and the 4vs4-SIMDop performs best.

We also investigated the influence of the actual SIMD version on the performance of our SIMDop. Figure 10 shows the results measured for an AVX version and an AVX512 implementation. The AVX512 implementation is twice as fast as the AVX due to the width of the AVX512 registers.

And finally, we evaluated the performance gain using SIMD optimized version of simultaneous BVH traversal compared with non-optimized version (see Figure 11). We roughly get a speedup around 2 for both 4vs4-SIMDop and 1vs16-SIMDop version, which is below our expectation since the AVX512 register can process 16 data at one time.

Hence, we investigated further by using a profiling tool Intel® VTune™ to profile the actual collision query timing. Table II shows profiling result for object Nao using 4vs4-

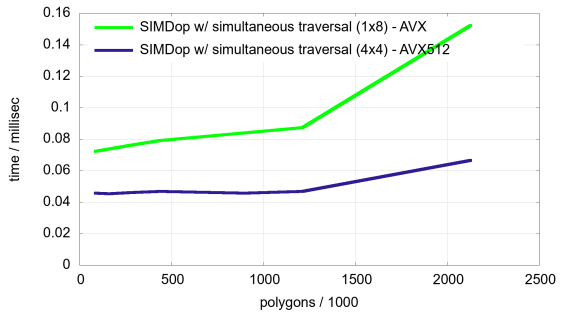


Fig. 10: Average collision query times for AVX implementation of our SIMDop in comparison with AVX512.

SIMDop. According to VTune, our 4vs4-SIMDop is able to vectorize 72.70% floating point instructions with the full vector capacity, which should theoretically give us a speedup of $\frac{72.7\times16}{100}$, however the gain is bound by memory, which took around 40.3% of computing time (whereas 33.40% of the time is stalled by main memory access). And also, our 4vs4-SIMDop has to test more orientations as much as three times more on average compared with non-optimized version (see Figure 12). In the end, we get roughly a speedup of 2 for the SIMD optimized version compared with the non-optimized version.

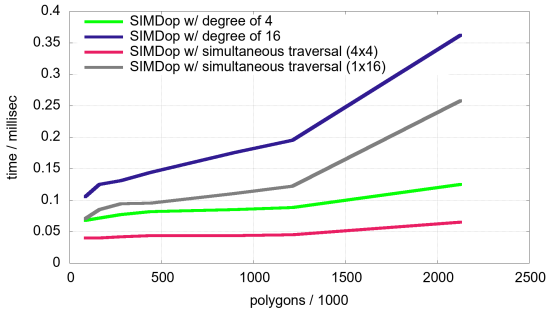


Fig. 11: Average collision query times using object Nao for our SIMDop with and without SIMD optimized simultaneous traversal.

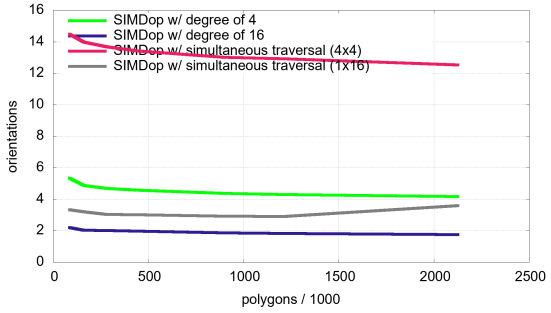


Fig. 12: Average orientations tested for a single bounding volume test of our SIMDop with and without SIMD optimited simultaneous traversal.

	SIMDop w/ degree of 4	SIMDop 4vs4
L1 Bound	3.00%	2.00%
L2 Bound	1.20%	1.60%
L3 Bound	2.70%	3.30%
DRAM Bound	21.10%	33.40%
Floating Point Vectorization	0%	72.70%

TABLE II: A performance analysis of our 4vs4-SIMDop using Intel® VTune™ for Nao.

VI. Conclusions And Future Work

We have presented two versions for a SIMD optimized bounding volume hierarchy for simultaneous BVH traversal. The main idea is to use higher n-ary trees instead of classical binary trees. We have presented several new heuristics for the top-down construction of such tree data structures with higher branching factor.

The BNG-based method performs best. Even if we tested only up to 16-ary trees, the clustering-based construction is already prepared to support higher branching factors following future SIMD developments. Our results show that, depending on the object, our SIMDop BVHs outperform traditional BVHs by more than an order of magnitude.

Our approach also opens up several directions for future work. For instance, we would like to include magnification control to the BNG construction algorithm. Moreover, other clustering algorithms than BNG could be considered. In this work, we relied on DOPs as BVs because of a fair comparison with the manual optimized traversal scheme. However, we would like to investigate also other BV types that do not have the problem of the later escape of the for-loop. Also the influence of the number of orientations for the DOPs requires further investigations. Finally, probably other applications using BVHs like ray tracing or occlusion computations could benefit from our SIMDop BVHs, too.

Acknowledgment

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

[1] M. Reggiani, M. Mazzoli, and S. Caselli, “An experimental evaluation of collision detection packages for robot motion planning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, Sept 2002, pp. 2329–2334 vol.3.

[2] G. Zachmann, “Rapid collision detection by dynamically aligned dop-trees,” in *Proceedings of the Virtual Reality Annual International Symposium*, ser. VRAIS ’98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 90–. [Online]. Available: http://dl.acm.org/citation.cfm?id=522258.836122

[3] S. Trenkel, R. Weller, and G. Zachmann, “A benchmarking suite for static collision detection algorithms,” in *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, V. Skala, Ed. Plzen, Czech Republic: Union Agency, 29 January–1 February 2007. [Online]. Available: http://cg.in.tu-clausthal.de/research/colldet_benchmark

[4] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Trans. Graph.*, vol. 15, no. 3, pp. 179–210, 1996.

[5] G. van den Bergen, "Efficient collision detection of complex deformable models using aabb trees," *J. Graph. Tools*, vol. 2, no. 4, pp. 1–13, Jan. 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=763345.763346

[6] G. Zachmann, "Minimal hierarchical collision detection," in *Proceedings of the ACM symposium on Virtual reality software and technology*, ser. VRST '02. New York, NY, USA: ACM, 2002, pp. 121–128. [Online]. Available: http://doi.acm.org/10.1145/585740.585761

[7] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, Jan. 1998. [Online]. Available: http://dx.doi.org/10.1109/2945.675649

[8] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: a hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 171–180. [Online]. Available: http://doi.acm.org/10.1145/237170.237244

[9] S. A. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition," *Computer Graphics Forum (Proc. of EUROGRAPHICS 2001)*, vol. 20, no. 3, pp. 500–510, 2001.

[10] S. Krishnan, A. Pattekar, M. C. Lin, and D. Manocha, "Spherical shell: a higher order bounding volume for fast proximity queries," in *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, ser. WAFR '98. Natick, MA, USA: A. K. Peters, Ltd., 1998, pp. 177–190. [Online]. Available: http://dl.acm.org/citation.cfm?id=298960.299006

[11] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Nov. 14 1999. [Online]. Available: http://citeseer.ist.psu.edu/408975.html;ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/ssv.ps

[12] T. Larsson and T. Akenine-Möller, "Bounding volume hierarchies of slab cut balls." *Comput. Graph. Forum*, vol. 28, no. 8, pp. 2379–2395, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/cgf/cgf28.html#LarssonA09

[13] T. He, "Fast collision detection using quospo trees," in *Proceedings of the 1999 symposium on Interactive 3D graphics*, ser. I3D '99. New York, NY, USA: ACM, 1999, pp. 55–62. [Online]. Available: http://doi.acm.org/10.1145/300523.300529

[14] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar, "Rapid and accurate contact determination between spline models using shelltrees," 1998.

[15] U. Schwesinger, R. Siegwart, and P. Furgale, "Fast collision detection through bounding volume hierarchies in workspace-time space for sampling-based motion planners," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 63–68.

[16] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3859–3866.

[17] D. Ferguson, M. Darms, C. Urmson, and S. Kolski, "Detection, prediction, and avoidance of dynamic obstacles in urban environments," in *2008 IEEE Intelligent Vehicles Symposium*, June 2008, pp. 1149–1154.

[18] N. Roussopoulos and D. Leifker, "Direct spatial search on pictorial databases using packed r-trees," in *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '85. New York, NY, USA: ACM, 1985, pp. 17–31. [Online]. Available: http://doi.acm.org/10.1145/318898.318900

[19] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez, "Str: A simple and efficient algorithm for r-tree packing," Institute for Computer Applications in Science and Engineering (ICASE), Tech. Rep., 1997.

[20] J. Goldsmith and J. Salmon, "Automatic creation of object hierarchies for ray tracing," *IEEE Comput. Graph. Appl.*, vol. 7, no. 5, pp. 14–20, May 1987. [Online]. Available: http://dx.doi.org/10.1109/MCG.1987.276983

[21] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in o(n log n)," *Symposium on Interactive Ray Tracing*, vol. 0, pp. 61–69, 2006.

[22] I. Wald, "On fast construction of sah-based bounding volume hierarchies," in *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, ser. RT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 33–40. [Online]. Available: http://dx.doi.org/10.1109/RT.2007.4342588

[23] C. Lauterbach, M. Garland, S. Sengupta, D. P. Luebke, and D. Manocha, "Fast bvh construction on gpus." *Computer Graphics Forum*, vol. 28, no. 2, pp. 375–384, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/cgf/cgf28.html#LauterbachGSLM09

[24] G. Zachmann, "Virtual reality in assembly simulation – collision detection, simulation algorithms, and interaction techniques," Dissertation, Darmstadt University of Technology, Germany, May 2000.

[25] G. Zachmann and E. Langetepe, "Geometric data structures for computer graphics," in *Proc. of ACM SIGGRAPH*. ACM Transactions of Graphics, 27–31July 2003. [Online]. Available: http://www.gabrielzachmann.org/

[26] J. Mezger, S. Kimmerle, and O. Etzmuß, "Hierarchical Techniques in Collision Detection for Cloth Animation," *Journal of WSCG*, vol. 11, no. 2, pp. 322–329, 2003.

[27] P. Agarwal, L. Guibas, A. Nguyen, D. Russel, and L. Zhang, "Collision detection for deforming necklaces," *Computational Geometry: Theory and Applications*, vol. 28, pp. 137–163, 2004.

[28] R. Weller, D. Mainzer, A. Srinivas, M. Teschner, and G. Zachmann, "Massively parallel batch neural gas for bounding volume hierarchy construction," in *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Bremen, Germany: Eurographics Association, Sept. 2014.

[29] M. Reggiani, M. Mazzoli, and S. Caselli, "An experimental evaluation of collision detection packages for robot motion planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, Sep. 2002, pp. 2329–2334 vol.3.

The 25th International Conference on 3D Web Technology (Web3D '20)
Virtual Event, Republic of Korea, November 9–13, 2020

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

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.

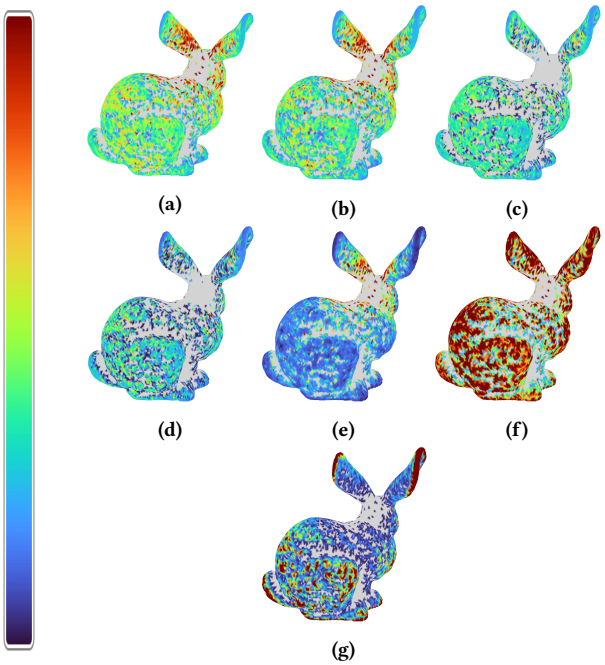


Figure 1: An example of heatmap based on configuration's (a) average timings, (b) median timings, (c) standard deviation timings, (d) median absolute deviation timings, and (e) min timings, (f) max timings, and (g) density.

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.

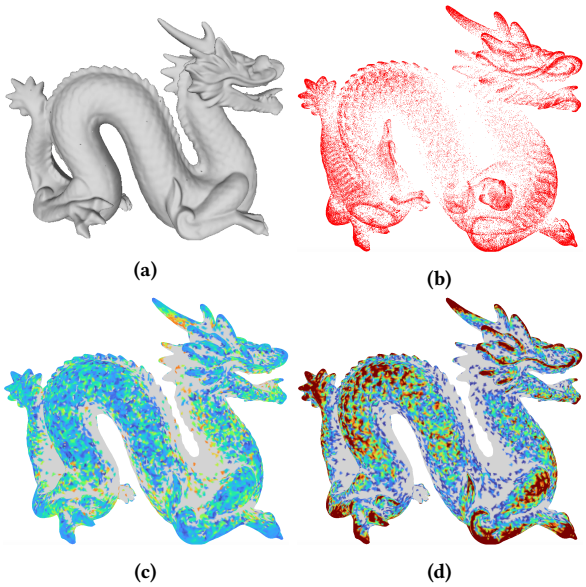


Figure 2: Heatmap generation pipeline based on benchmark's result: (a) 3D object, (b) closest points of all configurations, (c) generated heatmap based on algorithm timings, and (d) generated heatmap based on configurations density.

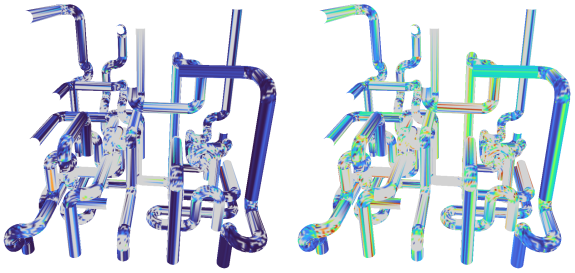
(2) Standard Deviation (Figure 1c) and Median Absolute Deviation (Figure 1d)
   - to visualize outlier regions where algorithm's timing could differs greatly between slightly different configurations.
(3) Configuration density (Figure 1g)
   - to visualize regions that are extensively or hardly checked 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

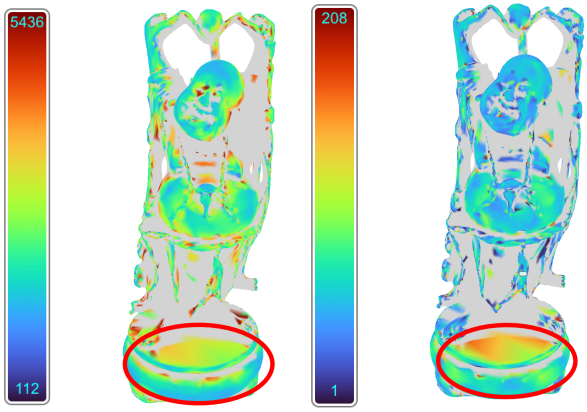
A primary goal of OpenCollBench as a benchmark as a service is to simplify the time-consuming process of integrating CD and configuring algorithms and to provide a common hard- and software platform to produce long-term reproducible and comparable results. We have realized this by a web-based client-server 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



(a) (b)

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.



(a) (b)

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.

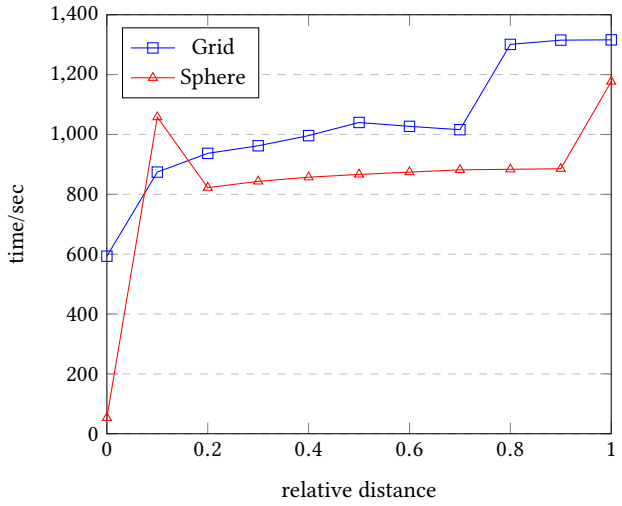


Figure 5: Time needed to generate around 200k configurations based on *Grid* and *Sphere* position finding methods for object bunny with 65k polygons at various predefined distances. The position finding methods tend to be slower at the higher distance between objects.

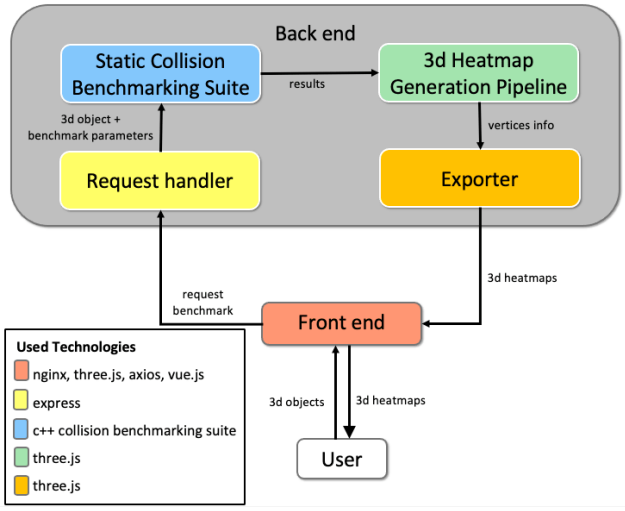


Figure 6: System Overview of OpenCollBench, which consists of two parts, namely *front end* that enable user to upload 3d object and select benchmarking parameters, and *back end* that process incoming benchmarking request and return heatmap as result.

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:

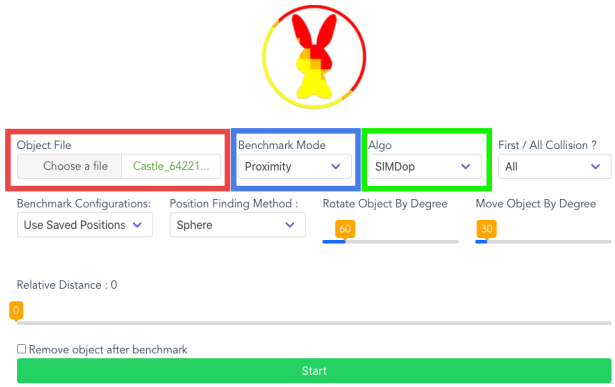


Figure 7: Interactive Graphical User interface (GUI) for OpenCollBench, which enable user to upload object (red box) and selecting benchmark parameters interactively. The option panels connected with each other, i.e. changing *Bench Mode* (blue box) to proximity will display algorithms that support promxity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

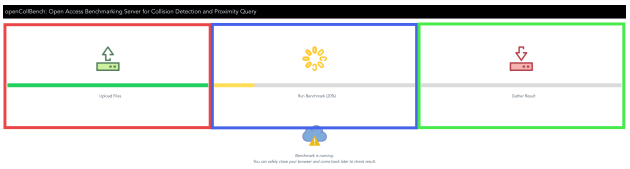


Figure 8: Benchmark's progress GUI for OpenCollBench, which consists of three parts, namely, *left* (red box) showing progress of object uploaded by user, *middle* (blue box) showing benchmarking progress including configurations generation, and *right* (green box) showing progress of heatmap generation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

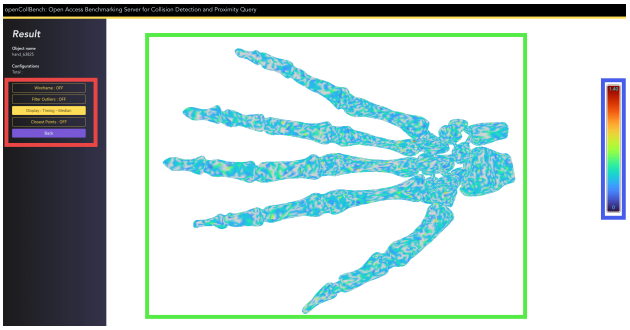


Figure 9: Benchmark's results GUI for OpenCollBench, which showing generated heatmap based on benchmarking results. Left panel (red box) enable user to select different mapping value, middle panel (green box) showing generated heatmap, and right panel (blue box) showing mapping color value.

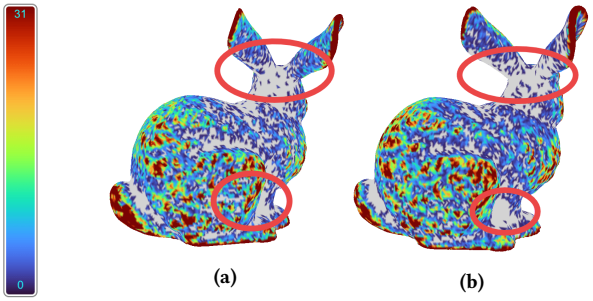


Figure 10: Heatmaps of object bunny with around 65k polygons based on configuration's density of around 200k configuration using position finding method (a) Grid, and (b) Sphere at a relative distance of 0.0. The *Grid* method is able to generate more configurations at concave area (red circles) compared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

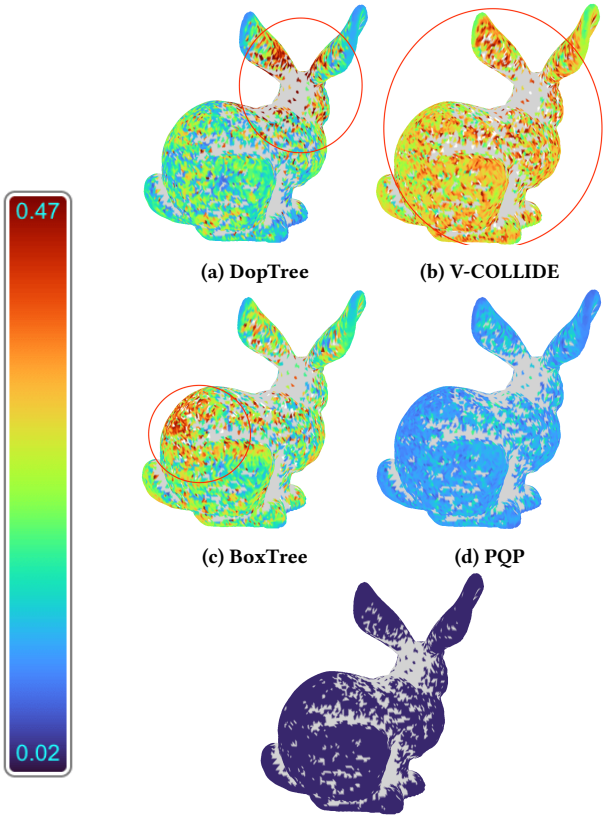
http://opencollbench.com. Currently, only files in the OBJ format can be uploaded by the user. In general, the benchmarking suite and three.js support a wide variety of different 3D file formats; however, they have to be integrated manually into three.js. For this reason, but also for security reasons, 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.

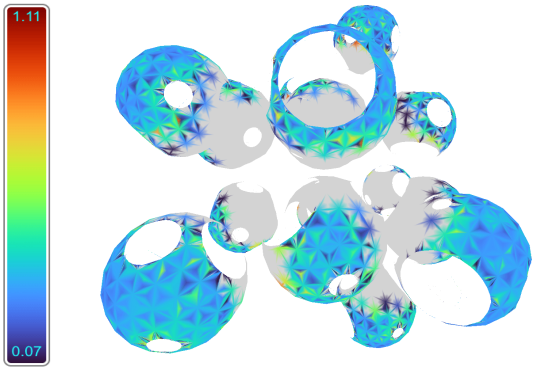


Figure 12: Heatmap result of object schwamm with 95k polygons based on median absolute deviation (timing in milisec) using DOPTree. The outer region does not fluctuate much, whereas the inner region fluctuates up to 1.1 milisec between slightly different configurations.

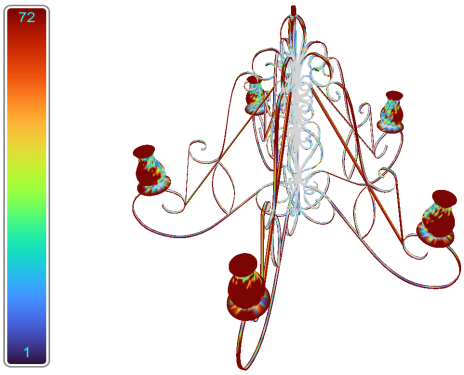
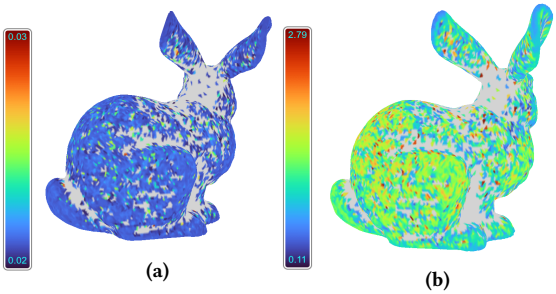


Figure 13: Heatmap of object lustre with 120k polygons based configuration's density of 200k configurations generated using *Grid* method after removing outlier. The inner region rarely checked by algorithms, whereas the outer region heavily tested.



(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.

The 18th Gesellschaft für Informatik e.V. (GI) workshop on the Virtual Reality and Augmented Reality (VR/AR)
Sankt Augustin, Germany, September 9-10, 2021

Grasping for reality - How can we improve the digital representation of human grasp behaviour?

Janis Rosskamp, Toni Tan, René Weller, Gabriel Zachmann

University of Bremen
Germany

Abstract:

In this paper, we present a heat- and forcemap representation of human grasps of objects in virtual reality (VR). For that, we divided the texture of the object into cells and depending on the number of detected contacts the texture is colored appropriately. The same method is used to visualize the force the hand applies on a particular spot on the object. While objects are manipulated, we also detect automatically from a list of over 30 grasp types how the object is grasped. Due to these aspects, our application provides a new and helpful way to represent hand grasps in a digital way which can be used in a wide array of possibilities. The knowledge on how humans grasp is an important topic for applications in robotics or prosthetics and would benefit the digital representation of the hand in general.

Keywords: grasp visualization, grasp type detection, heatmap, forcemap

1 Introduction

Advances in grasping algorithms and hand tracking methods enable interaction in virtual environments through natural grasping. This allows experiments in VR where participants can interact naturally with virtual objects. By performing experiment in VR, collecting information like causal and intuitive physics from the environment or contact surface during grasping are easier compared with real-life setup as well as repeating experiments using same setup. Subsequently, data obtained in these experiments can be used to gain insight into human grasps. This knowledge can then be applied to fields like robotic grasping where the amount of data needed to learn grasping task could exceeds what can be provided with reasonable effort [LPK+18]. Another field like prostheses, or even grasping in VR when only simple input devices are available could also be benefited.

Many factors influence human grasps which makes analyzing it a complex topic. To understand human grasping behavior, it is important to look at the interaction with many different objects. We need to recognize where and how objects are touched and held. Since what seems intuitive for humans is very complex in robotics, for instance. After all, how is a robot supposed to know that a cup is touched differently than a cereal box? Or, when grasping object, how much force should be applied? In VR, objects and manipulation scenarios can be added quite easily, which makes it a prime candidate for this investigation. In this paper, we present two different methods to gain knowledge during grasping operations.

The first method obtains contact points and forces between object and hand. These contact points are a great addition in visualizing the posture of a grasp and helpful to replicate grasps. They might even be used as training data for robotic grasping. Our method uses the approach proposed by Rosskamp et al. [RMW+21] for the generation of contact points and heatmap visualization. We improved their method by defining a grid on the object where each cell represents a possible grasping point. Color is assigned depending on how often a grid cell is in contact with a specific finger. Similar to the contact point heatmap, we added a force map that indicates how strong an object is grasped at which point. The applied force is a key variable when holding objects. If there is not enough force applied, the object could fall down and if it is too much, the object could crack. Using these heatmaps, it should be directly visible where and how forcefully one should touch certain objects to lift them. This could be used, for example, to train a robot to touch and hold objects correctly.

The second method discriminates between grasp types. This is a vital aspect of human grasp behavior analysis. Different objects require different handling to ensure proper treatment. Our application uses a combination of collision detection and rotation angle analysis to discriminate between over 33 grasp types.

2 Related Work

2.1 Heatmap

The use of heatmaps is a common way to visualize data on objects. For example, in marketing applications, eye tracking is used to investigate which parts of a website are particularly appealing or important [ULCP17]. For eye-tracking, Pfeiffer et al. [PM16] proposed a method to generate these heatmaps on 3D objects in real-time. Because eye-tracking data vastly differs from hand tracking, these results cannot be applied readily.

In [NFG19] human-object interaction is investigated on videos of humans performing tasks. Everyday objects are filmed during their usage and zones where the object was touched most often were identified and visualized with heatmaps.

In Taheri et al. [TGBT20] the contact points between hand and objects are visualized in heatmaps. The contact points are determined by grasping real objects. Both the hand and object are tracked using motion capturing and the collisions are computed on digital twins. Errors in tracking may lead to objects and hands either not touching or penetrating in the digital representation.

Liu et al. [LZX+19] developed a glove for hand tracking and used a caged-based grasp to manipulate objects in VR. The contact areas are visualized in heatmaps. While the caged-based approach is easy to set up, it lacks detailed collision and force calculations. This work mainly modifies the work of Rosskamp et al. [RMW+21] were heatmaps were created on objects to show which finger touched the object at what point. Their representation is not optimal, and sometimes no clear identification of the fingers on the object is possible. We improve their work by modifying the heatmap representation and adding forcemaps.

2.2 Grasp Type Detection

The Grasp Type Detection that was developed in the course of this work serves to detect the 33 grasp types defined in the GRASP Taxonomy [FRS+15], as it provides quite a complete overview of the most commonly used grasps.

Other Grasp Type Detection techniques use visual data as input. Guo et al. detect geometric shapes in photos to analyze 6 different grasps [GSF+17]. While the angles of the fingers need to be derived out of the input data, our hand tracking approach directly serves the values, so that there should be little possibility for conversion errors.

Kakoty and Hazarika use EMG data as input for their neural network detection of 6 grasps types [KH11]. As gathering EMG data requires assistance of medical trained technicians, we would not have been inclined to do it on our own, which was once again, one of the reasons for us to use hand tracking.

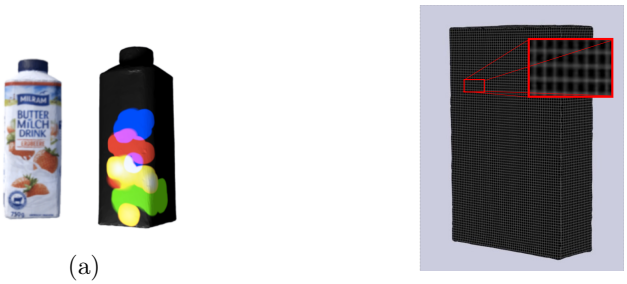
Many grasp type detection techniques, like Heumer et al. [HAWJ07] focus on detecting only six grasp types, following the Schlesinger taxonomy of hand grasps. However, this taxonomy is too simple to fulfil our application's use in the kitchen environment. For example, Grasp 19 *Distal Type* of GRASP Taxonomy is specified for using scissors, which we considered a commonly used grasp while cooking, such a grasp is rarely covered in other Grasp Type Detection techniques.

3 Grasp Visualization

In this section, the generation of heatmaps on objects for both contacts and forces is discussed. This is necessary, for instance, to understand human grasping behaviour, i.e., to recognize where and how objects are touched and held, or for robotic agent to learn how to grasp objects. For this, some necessary tools, i.e. hand tracking and collision detection, are needed. We are using the UnrealHaptics [RMW+21] framework with the Unreal Game Engine. This allows the easy integration of hand tracking with a Cyberglove, providing us a virtual representation of a hand. Additionally, UnrealHaptics allows custom collision detections. In our case, we utilize CollDet [Zac01], which uses an inner sphere tree to detect very detailed collisions between hand and object. Using CollDet has the advantage of not only providing contacts but forces as well.

3.1 Heatmaps

The approach in [RMW+21], checks the collision of the hand with the object in virtual reality and colors the object at this point. Here a distinction is made between the fingers where a different color is assigned to each finger. While the contact points are well represented and give a detailed insight into where the object is touched, the identification of finger is not yet optimal. This is obvious if the heatmap not only represents one but multiple grasps of the same object. As can be seen in Fig. 1a, colors of the different fingers mix as soon as two fingers touched the same spot so that it is not obvious which finger has touched this point.



(a) (b)

Figure 1: (a) Heatmap from [RMW+21] with the issue of colours mixing. (b) Visualization of the grid on the object used for the creation of heatmaps (zoom in red).

To optimize this approach, a new method for generating heatmaps was developed as part of this work. The idea is to construct a grid on the object's texture instead of object geometry. Ideally, the grid size should be as small as one pixel of object's texture. Figure 1b shows grid with size as small as one pixel of object's texture.

The grid is seen as an array of numbers, where each cell, i.e. each pixel, contains information about how often it was touched by which finger. From this information, it is then decided which is the *dominant finger* at that pixel, i.e. the finger that collided most often. Finally, the pixel is colored accordingly, with a color assigned to each finger. Unaffected points are not colored and are left black. This can be seen in Fig. 2. To find the point on the surface of the object, a ray the size of the collided sphere is cast onto the surface. This excludes the possibility of mixing colors and thus ensures clear identification of the fingers as the heatmap in Fig. 2 clearly shows.

Figure 2: The hand grasping the object on the left, the texture for the Heatmap in the middle and the final Heatmap on the right.

With this heatmap, it is now possible to analyze where different objects are touched and how often they are touched. The colors represent each finger, making it easy to see how the object was held.

3.2 Implementation: Forcemaps

Another important variable to understand human grasping behavior is the applied force when lifting different objects. After all, one could destroy a milk carton with too much force

applied while lifting it. To represent the applied force, a forcemap was developed as part of this work, showing at which point how much force had to be applied to the object.

The generation of the forcemap works similar to the previous heatmap with a grid whose cells reflect the pixels and are colored accordingly. Here, strong forces are displayed in red and weak forces in green, as shown in Fig. 3. Unaffected points are colored black. To enable a transition of the colors, the colors are assigned in steps of 10% of the force. Thus, orange and yellow represent a medium-strong force.

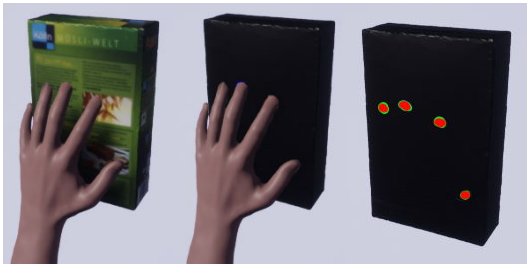


Figure 3: The hand grasping the object on the left, the texture for the forcemap in the middle and the final forcemap on the right.

3.2.1 Force Calculation

To calculate the force applied to the object we represent the real hand with two virtual hands. The first one is rendered and due to the collision detection cannot penetrate the object. The second hand represents the actual position of the fingers but is not rendered. The penetration depth of this hand is a measure for the applied force, i.e. larger penetrations are equivalent to stronger forces. In order to calculate the applied force, the pair of colliding spheres between object and hand in the inner sphere tree with their respective coordinates and radius is needed. The penetration depth x is calculated using the radius r_1, r_2 and the distance d with $x = \frac{r_1+r_2-d}{2}$ which is illustrated in Fig 4.

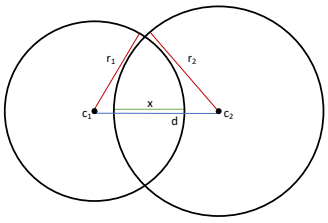


Figure 4: Illustration of the force calculation principle.

4 Grasp Type Detection

The grasp that needs to be applied depends heavily on the objects that should be picked up. For performing automatic grasping, it is, therefore, useful to identify which kind of grasp is

in use for a certain kind of object. While a water bottle tends to be gripped with curved fingers, we hold a breakfast board in an outstretched position (see Figure 5). In a future scenario, this information could be used the other way around. Having learned what grasp is applied for which object, a robot could grasp any object the appropriate way, avoiding it to fall down or break.

The GRASP Taxonomy [LFNP14] provided an overview of the hand grasps we wanted to detect within the kitchen use. It contains 33 grasp types categorized by thumb orientation i.e adducted or abducted. In this paper, we further subdivided the grasps into 17 categories so they would be easier to work on as grasps in every category are fairly close to each other. Within a category the same type of grasp is depicted, only the shapes and sizes of the objects vary. GRASP Taxonomy also considers whether arm movement is required performing the grasp [FRS+15]. One of the primary reasons behind working on top of GRASP Taxonomy was the number of details accounted for in it. Aside from simple curvy or straight grasps, there is a high variation between fingers in use. Additionally, very specific items, such as scissors and chopsticks, are also accounted for. For applying our Grasp Type Detection in a kitchen environment later on this seems to fit perfectly. Especially having the option to easily extend the grasp set when introducing new objects, played a big role in choosing GRASP Taxonomy, because it is to be expected, that our object set will grow in the future and grasps might change due to that.

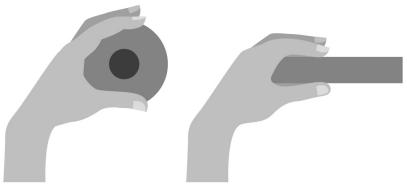


Figure 5: Grasping a water bottle would be different from grasping a cutting board

The first step in our Grasp Type Detection is finding out which fingers are active meaning which fingers are touching an object, the hand palm, or perhaps another finger, then narrowing down to the group of possible grasps. For this purpose, we come up with two approaches based on raycasting & collision detection.

4.1 Raycasting

The idea of raycasting is shooting rays out of certain points of the hand and detecting where those rays hit. Therefore you can detect whether a finger is touching an object, the hand palm or perhaps another finger. As shown in Figure 6 in our application we shoot 30 rays. Each of the 5 fingers has the following 6 rays:

- 1 ray from each joints (upper, middle, base) towards the palm,
- 1 ray on the side (left, right) shooting sideways,
- 1 ray on the very tip of the finger shooting upwards.

The rays facing the palm are detecting whether a full finger is touching an object, the rays on the sides detect whether an object lies between fingers and the rays on the tip are

used for differentiating between touching an object with the flat or pointed fingertip. These rays are used in various functions, some of these functions are following:

- *IsFingerNailTouching* shoots a ray from the tip in the direction parallel to the finger.
- *IsFingerRightSideTouching* and *IsFingerLeftSideTouching* check if the object is touching the finger from right or left side of the finger, respectively.
- *IsFingerFrontTouching* checks if the object is touching the finger from the front (the side facing the palm).
- *IsFullFingerTouching* casts three rays (from upper, middle and base joints) in the direction away from the finger
- *IsOnlyTipTouching* casts three rays (from upper, middle and base joints) in the direction away from the finger. Additionally *IsFingerNailTouching* is called. Either upper joint ray or *IsFingerNailTouching* have to return a hit in addition to middle and base joints not returning a hit.

4.2 Collision Detection

In this approach, we chose Inner Sphere Trees [RW09] to perform collision detection. Beside able to find which fingers are active, we could also get contact points, penetration depths and minimal distance as well.

The idea behind Inner Sphere Trees is to fill objects (in our case, the fingers and the kitchen item) densely using non-overlapping spheres and build a tree hierarchy on top of it. To fill the objects with spheres, ProtoSphere [RW10] is used, which is a GPU-Assisted algorithm that can pack a mesh with spheres.

Since detection which part of the finger is colliding with the object is vital for Grasp Type Detection, we had had to break the fingers into three parts. These divisions occur at the joints of the fingers (upper, middle, base) and hence, the mesh has been broken into these parts as well. Although, for the purpose of our paper, we only need to fill in the upper and the middle parts, as those are the only parts essential to the detection. A simplified version of the spheres can be seen in Figure 6.

As soon as those objects overlap, a contact is recognized. A function called *IsTouching* is called to check if a finger is actually touching the object. This function checks all available methods to detect whether the finger is touching the object from any direction (including the sides).

Rotation Angles Analysis The colliding fingers are already a good indicator for which grasp could have been applied, but to allocate a grasp more accurately, also the finger rotation angles need to be analysed. The input data for our detection comes from the motion capture glove *CyberGlove III*, from which we got the 15 sensor points, the rotation angles of base, middle and upper joint of each finger. In that way, the bone angles of the fingers can be measured directly using a highly reliable detection [HAWJ07].

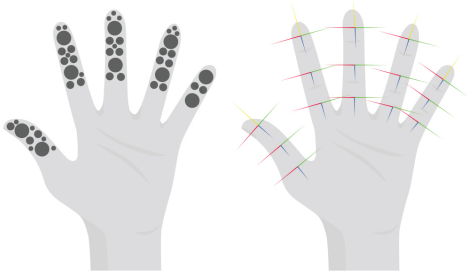


Figure 6: Left: Hand filled with Inner Sphere Trees. Right: Rays being cast from the nails, sides and joints of the hand.

Even without using Machine Learning, we can systematically determine the rotation values of certain grasps, by exploring thresholds from the finger positions. In the initial position all finger joints are straight and share the value 0. When turning towards the palm the numbers decreases to at most -100. Having those values as anchors, the values for other functions can be derived of that. Fingers are touching the palm when the base and middle joint are at -70.0 degrees or less and the upper joint would be alright, if it stayed straight. A bone is straight within the threshold of -15 to 15. A finger is straight when it's three bones are straight. A thumb is adjacent to the fingers (adducted) if the base of the thumb has a rotation angle of 20 degrees or more. If the distance between index and thumb is less than 0.3, they are touching, if it's less than 5 it's a rather small distance and also probably a rather small object being touched and for values bigger than that but lower than 10 it would be a rather big object. These thresholds can later be used for shortcutting the detection of specific grasps.

4.3 Grasp Type Detection

Our algorithm is an interpretation of the GRASP taxonomy. In general, we group the grasps by which fingers are in use and how similarly the hand is shaped. In contrast to the taxonomic approach we disgard the thumb position as a cluster, as well as the hand's opposition type. Therefore, we don't keep exactly the same groupings as the taxonomy, but most grasp categories still remain collectively. While the taxonomy contained 17 grasp categories, our algorithm distinguishes between 6 active finger combinations (see Appendix 7). The first step of the Grasp Type Detection is finding which finger is touching the object using collision detection. The decision tree in Appendix 7 is used for further discrimination due to the earlier introduced collision detection or rotation angle analysing functions.

While for grasp 15 *Fixed Hook* it is enough to detect it's unique finger combination (everyone except the thumb), the more similar grasps need a more detailed differentiation. To find grasp 22 *Parallel Extension*, all five fingers much be touching the object entirely. If the index finger is not straight, then the straightness of all the other opposing fingers gets

detected. All straight fingers would then already indicate grasp 22, in contrast to grasp 17 *Index Finger Extension*, where middle, ring and pinky finger are facing towards the palm.

Application Output Within our application the output shows for each performed grasp his number and term following the GRASP taxonomy, the category number we interpreted out of that, it's thumbs orientation as it was an important factor within the taxonomy and lastly which fingers are touching the object, as the main classifier in our algorithm.

5 Discussion

Our work is based upon GRASP Taxonomy which became a widely used standard for the detection of single hand grasps. By using this Taxonomy as the underlying concept of the Grasp Type Detection, we make sure to build upon one of the most recognized works in this area. The use of Heatmaps in both ways provides an easy and intuitive way to let the user know where the object was touched and with what amount of force. Due to the difference in color choice, both Heatmaps have a distinct look and can't be easily confused.

The Heatmap as a result of our work provides a more approachable illustration of the contact data. The object is covered in a pixel-wide grid and filled with an inner sphere tree for collision detection. Each contact casts the shape of the finger on the grid and calculates the dominant finger for each touched pixel of the grid. This allows us to clearly show which finger was the most dominant in a general area. Pixels are then colored accordingly showing which finger touched the object on that spot. The dominant finger also allows us to prohibit the mixing of colors which results in a more meaningful heatmap.

This approach delivers a clear representation of the dominant grasp areas that is beneficial for the concrete representation of the touched areas. An improved implementation based on factors mentioned in the discussion would result in an even more meaningful Heatmap. Following experiments would be helpful in this process as well as they would help to refine the Heatmap. The approach which determines a dominant finger for each pixel is thereby a new approach which results in a promising, supportive Heatmap.

The heatmap which was created for this paper is calculated using only the fingertips meaning other finger parts are omitted from this calculation. Under these circumstances, Heatmaps using the whole hand would most likely result in different Heatmaps. As this hasn't been thoroughly tested yet, it can't be stated what a Heatmap with all finger parts would look like. As our approach is tied to the shape of the individual object each new object has to be individually prepared by filling it with an inner sphere tree, exchanging the given texture with a black one and providing the pixel grid. These tasks could easily be automated to save time and minimize the room for human errors.

With the previously mentioned forcemap we introduced a new way of indicating force in a digital way. Not only the calculation but the visualization is a complete new approach. The forcemap aims to be a helpful tool which indicates the amount of excessive force which was applied to certain areas of the object. Similar to the heatmap the whole object is covered in

a pixel-wide grid. The force is then calculated by measuring the intersection of the spheres that reside in the object and the hand. The penetration is the result of a calculation between overlapping spheres from hand and object. This can then be grouped by percentages from a small overlap to a near complete overlap. The calculated data will then be painted on the area of the forcemap to indicate the provided force.

Our way of calculating the force by measuring the penetration depth has not been compared with real grasps. It thereby just provides the way of calculating the penetration without indicating a correlation to real grasps. This means that our work on the Forcemap can't be compared to a measurement of force like Newton.

Our grasp type detection could find use in multiple areas that have needed a way to represent the hand but have struggled due to the complexity of the task. Our method provides an intuitive way that can be run in real time with a reasonably sized dataset and aims to be a one-does-all application. While the detections of related work from Guo et al. [GSF+17] or Kakoty and Hazarika [KH11] focus on six grasps, ours can detect over 30 grasps types, making it specific enough to handle any kitchen item with proper care. For discriminating between the grasps raycasting paid off for detecting collision, in contrast to the inner sphere trees it was not in need of heavy computation while still serving our purpose. For our application the accuracy of the inner sphere trees has not been necessary, furthermore the intuitive integration of more rays has not been served when adding further inner sphere tree colliders.

6 Conclusions & Future work

We have presented an automatic grasp type detection for single hand grasps based on a decision tree and able to recognize all grasp types from GRASP taxonomy.

Our approach opens up several directions for future work. For instance, extending the grasp type detection by stating what kind of object was interacted with. This can be done simply by creating a database of possible objects, in the case of a limited study and application, or it can be achieved by checking the shape of collision detection volumes on the object that is being touched. Also, more data should be collected for refining the grasping algorithm and evaluating it extensively.

Moreover, future research could focus on generating heatmaps with all finger parts in place. This would result in new heatmaps revealing further information about contact points. The difference in meaningfulness between these new heatmaps and our approach could be analyzed as well. Following experiments can additionally analyze how meaningful the heatmaps become after multiple experiments.

Experiments with forcemap would have to be conducted to prove a significant importance of the forcemap and test if the penetration depth is expressive and comparable with units of measurement like Newton. Further insights lie in the comparison of the force application when lifting the same objects in a virtual reality and the real world. Next steps would be a study with different objects to get heatmaps that can be analyzed, compared and discussed.

Literatur

[FRS+15] Thomas Feix, Javier Romero, Heinz-Bodo Schmiedmayer, Aaron M Dollar, and Danica Kragic. The grasp taxonomy of human grasp types. *IEEE Transactions on human-machine systems*, 46(1):66–77, 2015.

[GSF+17] Di Guo, Fuchun Sun, Bin Fang, Chao Yang, and Ning Xi. Robotic grasping using visual and tactile sensing. *Information Sciences*, 417:274–286, 2017.

[HAWJ07] G. Heumer, H. B. Amor, M. Weber, and B. Jung. Grasp recognition with uncalibrated data gloves - a comparison of classification methods. In *2007 IEEE Virtual Reality Conference*, pages 19–26, 2007.

[KH11] N. M. Kakoty and S. M. Hazarika. Recognition of grasp types through principal components of dwt based emg features. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–6, 2011.

[LFNP14] Jia Liu, Fangxiaoyu Feng, Yuzuko C Nakamura, and Nancy S Pollard. A taxonomy of everyday grasps in action. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 573–580. IEEE, 2014.

[LPK+18] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[LZX+19] Hangxin Liu, Zhenliang Zhang, Xu Xie, Yixin Zhu, Yue Liu, Yongtian Wang, and Song-Chun Zhu. High-fidelity grasping in virtual reality using a glove-based system. In *2019 international conference on robotics and automation (icra)*, pages 5180–5186. IEEE, 2019.

[NFG19] Tushar Nagarajan, Christoph Feichtenhofer, and Kristen Grauman. Grounded human-object interaction hotspots from video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8688–8697, 2019.

[PM16] Thies Pfeiffer and Cem Memili. Model-based real-time visualization of realistic three-dimensional heat maps for mobile eye tracking and eye tracking in virtual reality. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, pages 95–102, 2016.

[RMW+21] Janis Roßkamp, Hermann Meißenhelter, Rene Weller, Marc Rüdel, Johannes Ganser, and Gabriel Zachmann. UnrealHaptics: Plugins for Advanced VR Interactions in Modern Game Engines. *Frontiers in Virtual Reality*, 2:640470, April 2021.

[RW09] Gabriel Zachmann Rene Weller. Inner sphere trees for proximity and penetration queries. *Robotics Science and Systems (RSS)*, 2009.

[RW10] Gabriel Zachmann Rene Weller. Protosphere: A gpu-assisted prototype guided sphere packing algorithm for arbitrary objects. In *ACM SIGGRAPH Asia*, 2010.

[TGBT20] Omid Taheri, Nima Ghorbani, Michael J Black, and Dimitrios Tzionas. Grab: A dataset of whole-body human grasping of objects. In *European Conference on Computer Vision*, pages 581–600. Springer, 2020.

[ULCP17] Florina Ungureanu, Robert Gabriel Lupu, Adrian Cadar, and Adrian Prodan. Neuromarketing and visual attention study using eye tracking techniques. In *2017 21st international conference on system theory, control and computing (ICSTCC)*, pages 553–557. IEEE, 2017.

[Zac01] Gabriel Zachmann. Optimizing the collision detection pipeline. In *Procedings of the First International Game Technology Conference (GTEC)*, 2001.

Appendix

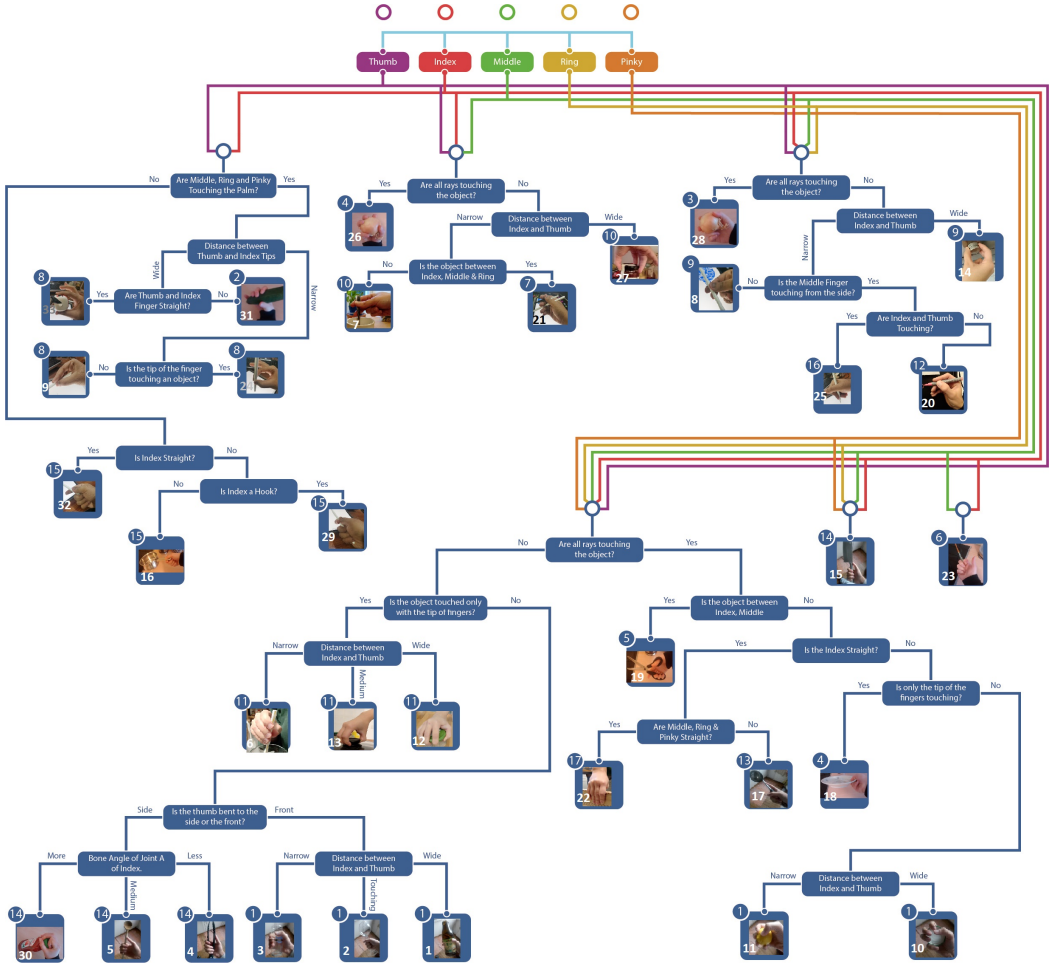


Figure 7: Our decision tree for automatic grasp type detection base on GRASP Taxonomy for kitchen use

The 27th International Conference on 3D Web Technology (Web3D '22)
Evry-Courcouronnes, France, November 2–4, 2022

A Framework for Safe Execution of User-Uploaded Algorithms

Toni Tan
toni@cs.uni-bremen.de
University of Bremen
Germany

René Weller
weller@cs.uni-bremen.de
University of Bremen
Germany

Gabriel Zachmann
zach@cs.uni-bremen.de
University of Bremen
Germany

ABSTRACT

In recent years, a trend has existed for an open benchmark aiming for reproducible and comparable benchmarking results. The best reproducibility can be achieved when performing the benchmarks in the same hard- and software environment. This can be offered as a web service. One challenge of such a web service is the integration of new algorithms into the existing benchmarking tool due to security concerns. In this paper, we present a framework that allows the safe execution of user-uploaded algorithms in such a benchmark-as-a-service web tool. To guarantee security as well as reproducibility and comparability of the service, we extend an existing system architecture to allow the execution of user-uploaded algorithms in a virtualization environment. Our results show that although the results from the virtualization environment are slightly slower by around 3.7% to 4.7% compared with the native environment, the results are consistent across all scenarios with different algorithms, object shapes, and object complexity. Moreover, we have automated the entire process from turning on/off a virtual machine, starting benchmark with intended parameters to communicating with the backend server when the benchmark has finished. Our implementation is based on Microsoft Hyper-V that allows us to benchmark algorithms that use *Single Instruction, Multiple Data (SIMD)* instruction sets as well as access to the *Graphics Processing Unit (GPU)*.

CCS CONCEPTS

• **Computing methodologies → Collision detection**.

KEYWORDS

benchmark as web-service, open benchmark

ACM Reference Format:
Toni Tan, René Weller, and Gabriel Zachmann. 2022. A Framework for Safe Execution of User-Uploaded Algorithms. In *The 27th International Conference on 3D Web Technology (Web3D '22), November 2–4, 2022, Evry-Courcouronnes, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3564533.3564560

1 INTRODUCTION

In a computer-based application like collision detection or object detection, the benchmark is essential to measure the effectiveness and efficiency of proposed algorithms. Unlike object detection, which focuses on algorithms' accuracy, collision detection, on the other hand, focuses on both accuracy and performance. In object detection, only dataset and ground truth are needed for benchmarking, which makes it relatively non-problematic when it comes to benchmarking. On the other hand, since the existing benchmarking tools for collision detection are usually available as a standalone program [Trenkel et al. 2007][Woulfe and Manzke 2009][Wang et al. 2021][Weller et al. 2010], it needs to integrate the proposed algorithms into existing benchmarking tools, which could be problematic as the implementation, for instance, into bullet [Woulfe and Manzke 2009] is not always easy. Besides that, we might need to integrate existing algorithms that we want to compare with, as in most cases, they are probably not integrated yet into existing benchmarking tools. Re-implementing existing algorithms might not always be easy as different optimization could influence the performance. Besides that, the user will need to get used to the benchmarking programs, which is not always easy due to the complicated benchmark parameters. Not to mention the hardware constraints while benchmarking algorithms that use special hardware, i.e., *Advanced Vector Extensions (AVX-512)*.

An attempt to solve this is to offer benchmarking tools as web-service that is based on the benchmark proposed by [Trenkel et al. 2007]. Such an online service was proposed in [Tan et al. 2020] and can be accessed online at http://opencollbench.com. It allows users to choose between a set of pre-defined geometries or even upload their own 3D objects and compare the performance of different built-in collision detection algorithms. The results can be visualized in easy-to-understand diagrams.

In this paper, we extend the framework to allow users also to upload their own collision detection libraries and benchmark them against competitors directly and anonymously. However, this, on the other hand, becomes a security concern due to running unknown code. In order to guarantee the security of such a scenario, we shift the execution of user-uploaded algorithms into a virtualization environment. Additionally, we have implemented *WebSocket* as a communication protocol to communicate with the *Virtual Machine (VM)* when the benchmark process is finished.

Moreover, the entire benchmarking process, from turning on/off VM, starting benchmark with intended parameters, to communicating with the backend server, is automated within our framework. Our implementation is based on Microsoft Hyper-V, which allows us to create VM that support SIMD Instruction Sets and use GPU passthrough to access the GPU of the host system directly. This allows us to support algorithms that make use of SIMD Instruction Sets and GPU, such as SIMDop [Tan et al. 2019].

Web3D '22, November 2–4, 2022, Evry-Courcouronnes, France

ACM ISBN 978-1-4503-9914-2/22/11…$15.00
https://doi.org/10.1145/3564533.3564560

2 RELATED WORK

In computer-based applications, the process and goal of benchmarking vary across fields, i.e., object detection compared algorithms accuracy against ground truth annotated by a human [Tu et al. 2022]. Ray tracing compared the accuracy and running time by benchmarking against a set of predefined scenarios, e.g., the Benchmark for Animated Ray Tracing (BART) [Lext et al. 2001]. The proposed algorithms are evaluated against existing algorithms in terms of accuracy and running time in collision detection.

Benchmarking tools are typically provided as standalone programs. This is sufficient in object detection or ray tracing due to the simplicity of benchmarking results needed. On the other hand, collision detection is more complicated since the algorithms running time could be influenced by object shapes, object complexity, and even relative distance or rotation between objects. Usually, authors define a specific scenario on their own to test their algorithms [Otaduy and Lin 2003; van den Bergen 1998]. A comparison between algorithms could yield different results under a different scenario. There exist attempts to generalize the benchmarking procedure especially for rigid bodies collision detection [Diktas and Sahiner 2008; Trenkel et al. 2007; Weller et al. 2010]. Moreover, in some cases, authors do not make their proposed algorithms publicly available. An attempt to reinvent the algorithms could yield a different running time due to optimization. Not to mention the availability of hardware like SIMD Instructions Sets or GPU could make it impossible to benchmark algorithms like SIMDop [Tan et al. 2019].

An attempt to solve this is to offer benchmarking tool as web service [Tan et al. 2020].

3 OUR FRAMEWORK

Running unknown user-uploaded algorithms will always pose a risk, i.e., *Remote Code Execution (RCE)*. Directly analyzing and validating the code is not trivial, not to mention authors might not want to disclose their algorithms in some cases. Hence, it does make sense to run user-uploaded algorithms in case of doubt in an environment where it can not cause any damage. This could be done on another physical computer accessible over the network and that does not have access to critical systems and does not contain sensitive data. However, the fact that the machine is connected to other computers in a network is already a risk. It is also challenging to identify whether this system is compromised.

This is where the use of hardware virtualization comes in handy. Here, access to the physical machine's hardware is regulated by a so-called *hypervisor*. This can be an *Operating System (OS)* that runs natively on the hardware (Type 1), e.g., *Microsoft Hyper-V, VMWare ESXi* or software that runs on an operating system and simulates hardware access (Type 2), e.g., *Microsoft Virtual PC, Oracle Virtual Box, VMware Workstation*. A *virtual machine (VM)* can be started via the hypervisor, which operates completely isolated from the underlying systems. In addition, a virtual network can be configured with the hypervisor, to which only the host system and the virtual operating system have access. This means the virtual system has no access to external networks to which the host system is connected. On top of that, many hypervisor implementations offer a so-called *Snapshot* function that can save the state of the virtual machine at a specific point in time and restore it if necessary. This resets all data changed over the runtime, both on the virtual storage medium and the data in the virtual main memory. Since the backend server of *OpenCollBench* is running under windows, we chose to use Microsoft Hyper-V to implement our framework. This also has another advantage, as the virtualization API can be easily accessed using *Windows PowerShell*, i.e., getting the IP address of VM, creating new VM, turning on/off VM, and resetting VM, which makes it convenient for automating the benchmarking process in VM.

In this paper, we extend the capabilities of OpenCollBench to allow benchmarking of user-uploaded algorithms into a virtualization environment. Currently, new algorithms must be integrated by the administrator, which can be problematic in work-in-progress developments or due to non-disclosure agreements.

With our framework, it is sufficient for users to compile and upload their proposed algorithms as wrapper *Dynamic-Link Library (DLL)* specified by *OpenCollBench*. This keeps the user-uploaded algorithms confidential, and results are comparable with other publicly available algorithms as well as easy-to-understand visual diagrams provided by *OpenCollBench*.

3.1 Benchmarking in VM

In order to automate the process of running benchmarking tools in VM, we have implemented an additional service to listen for an incoming connection from the backend server and, on request, to start the benchmarking with the supplied parameters. The benchmarking result will be sent back to the backend server when finished. Since the benchmarking can take several minutes, a typical *Hypertext Transfer Protocol (HTTP)* connections would time out without a response from the server for such a long time. Web sockets [Melnikov and Fette 2011] are an ideal solution to this problem. They make it possible to establish and maintain a bidirectional connection between a client, in this case, the backend server, and a server. This is done via an initial handshake, which is still carried out using an HTTP-compatible protocol. After that, all data is transmitted in a binary protocol based on *Transmission Control Protocol (TCP)*.

In order to enable the WebSocket, which was started in the backend server, to communicate with the VM, it needs the *Internet Protocol (IP)* address of the VM, which can be queried using a PowerShell command (See A.1).

To ensure that the server is always running when the VM is started, a checkpoint was created with Hyper-V during operation. The VM is then always reset to this before it is started if necessary. This is done using a PowerShell script that is called from the backend server (See A.2). The caller thread then waits until the VM has started and the script terminates.

Currently, only 1 VM would be allowed to run at one time. This guarantees comparability and prevents malicious actors from overloading the benchmarking machine, i.e., by uploading malicious algorithms simultaneously (more or less) from different clients. In addition, the VM is always reset back to its initial state before starting a new benchmark job. This step prevents any system changes by either OS updates or malicious algorithms.

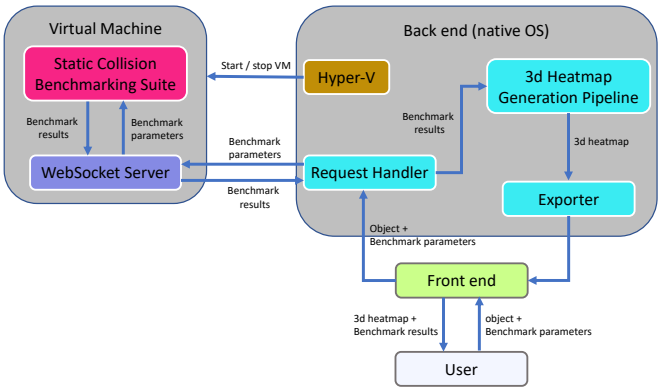


Figure 1: System Overview of our proposed framework, which extends the capability of openCollBench to benchmark user-uploaded algorithms in a secure virtualization environment

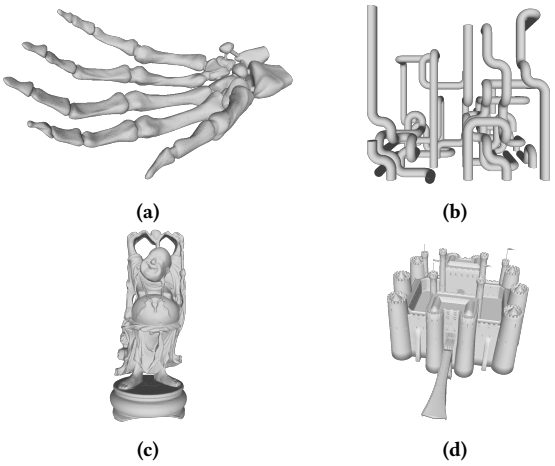


Figure 2: The objects we used in our timings: (a) hand, (b) pipes, (c) happy buddha, and (d) castle.

3.2 GPU Passthrough

Direct access to the host system's GPU is entirely feasible using Hyper-V and a full-fledged Windows VM. To do this, a so-called *GPU passthrough* must be configured under Hyper-V. The VM is given exclusive access to the GPU since the host system does not virtualize it.

For the configuration, the storage location of the desired graphics card must first be determined. To do this, the hardware can be selected via the Windows device manager and the property storage location paths can be selected under the *Details* tab. The storage location can be found in the first line of the text field. The command A.3 can be used to disconnect the graphics card from the host system via PowerShell.

However, the host system's GPU may stop working at this step if the host system only has one card available. To then assign the GPU to the VM, another command A.4 is used in PowerShell.

The graphics card should then be found in the VM's device manager, enabling a CUDA installation, thus able to run algorithms that make use of GPU computation.

4 RESULTS

We have implemented our framework based on Microsoft Hyper-V™. The automation and additional services have been implemented using *PowerShell* and *node.js*. As a result, we extended the capability of *OpenCollBench* to allow the execution of user-uploaded algorithms securely. Figure 1 shows the architecture of the extended system.

In the new architecture, the benchmarking execution was decoupled from the operation of the backend server. This ensures, among other things, that the CPU access of this process is not interrupted when there is a high load in the backend, which means the benchmarking results will stay consistent.

In order to take a closer look at the influence of the VM when benchmarking user-uploaded algorithms, we compared the results from VM with the native system. Figure 3 shows a comparison of running time for benchmarks executed in native and virtualization environments for both object Castle (Figure 2d) and Happy Buddha (Figure 2c) with up to 120k polygons. Each object in our benchmark (See Figure 2) consists of up to 200k configurations. As a result, each benchmark takes up to 20 minutes to finish, with the average collision check between 0.7 to 6.7 milliseconds. The results from the virtualization environment are slightly higher, which is expected due to the virtualization layer. However, it remains consistent with delta between 3.7% to 4.7% across different algorithms, object shapes, and object complexity. Figure 2 shows objects we used to measure the effectiveness of benchmarking in VM. Between several benchmarks runs under the same parameters, there could be a slight deviation between their running time. In the native environment, the deviation is typically less than 0.1%. This is also the case in the virtualization environment.

It is well known that the windows operating system comes with lots of apps pre-installed. This could affect the running time during the benchmark. Hence, we also measured the influence of the number of core a system hat towards algorithms running time. Figure 4 shows the comparison of running times for object Pipes (Figure 2b) with up to 120k polygons using DopTree algorithms under the native and virtualization environment with one core, two cores, and three cores allocated. The results are as expected since DopTree only makes use of one core. Hence adding more core to the virtualization environment does not improve collision running time.

5 CONCLUSIONS AND FUTURE WORK

We have presented a benchmarking framework for the secure execution of user-uploaded algorithms in the virtualization environment. The goal is to allow web-based benchmarking tools to execute user-uploaded algorithms and, at the same time, provide a security guarantee while running unknown code. Our implementation is done on top of existing web-based benchmarking tools, namely *OpenCollBench*. Additionally, we provided automation for running

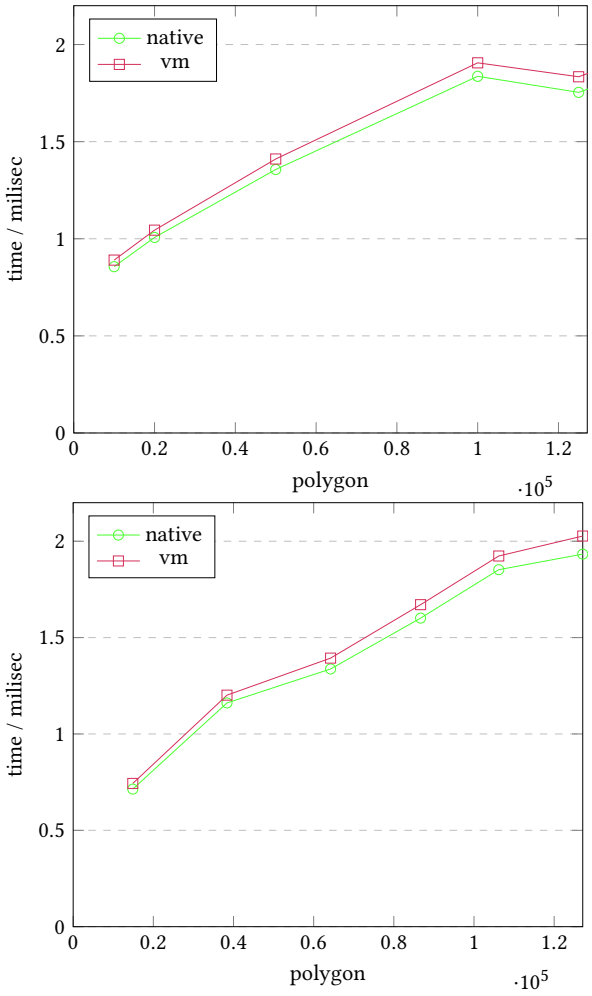


Figure 3: Average collision query time for the object Castle and happy buddha in native and virtualization environment. The delta are very similar for all objects.

benchmarking tools in the virtualization environment within our framework.

Our approach also offers interesting avenues for future work: for instance, by implementing the server endpoint as a REST endpoint, other services could also use the benchmarking server for example, to evaluate the proposed algorithm within a continuous integration pipeline when building an application. In this sense, a plugin for *integrated development environment (IDE)* such as *Visual Studio* would also be conceivable that allows the user to directly assess the effects of his changes to algorithms during development.

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(s) <R03> <A knowledge representation and reasoning framework for robot prospection in everyday activity>.

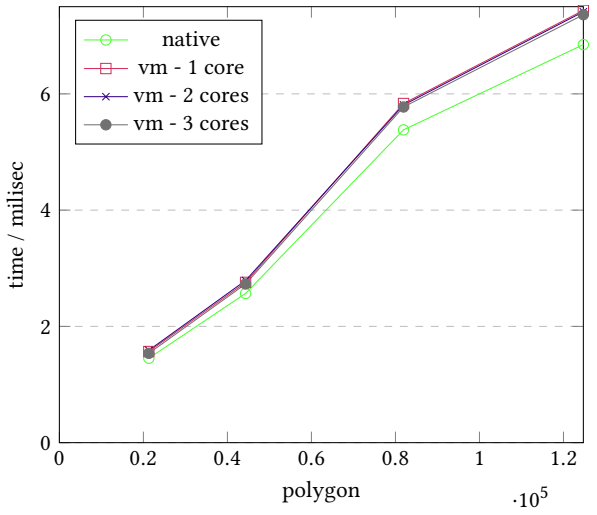


Figure 4: Average collision query time for the object Pipes in native and virtualization environment using 1 core, 2 cores, and 3 cores.

REFERENCES

Engin Deniz Diktas and Ali Vahit Sahiner. 2008. A Benchmarking Framework for Static Collision Detection. In *Theory and Practice of Computer Graphics*, Ik Soo Lim and Wen Tang (Eds.). The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/TPCG/TPCG08/107-113

J. Lext, U. Assarsson, and T. Moller. 2001. A Benchmark for Animated Ray Tracing. *IEEE Computer Graphics and Applications* 21, 2 (2001).

Alexey Melnikov and Ian Fette. 2011. The WebSocket Protocol. RFC 6455. https://doi.org/10.17487/RFC6455

Miguel A. Otaduy and Ming C. Lin. 2003. CLODs: Dual Hierarchies for Multiresolution Collision Detection. In *Eurographics Symposium on Geometry Processing*, Leif Kobbelt, Peter Schroeder, and Hugues Hoppe (Eds.). The Eurographics Association. https://doi.org/10.2312/SGP/SGP03/094-101

Toni Tan, René Weller, and Gabriel Zachmann. 2019. SIMDop: SIMD optimized Bounding Volume Hierarchies for Collision Detection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Macau, China, 7256–7263. https://doi.org/10.1109/IROS40897.2019.8968492

Toni Tan, Rene 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* (Virtual Event, Republic of Korea) *(Web3D '20)*. Association for Computing Machinery, New York, NY, USA, Article 9, 9 pages. https://doi.org/10.1145/3424616.3424712

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.

Zhengzheng Tu, Yan Ma, Zhun Li, Chenglong Li, Jieming Xu, and Yongtao Liu. 2022. RGBT Salient Object Detection: A Large-scale Dataset and Benchmark. *IEEE Transactions on Multimedia* (2022), 1–1. https://doi.org/10.1109/TMM.2022.3171688

Gino van den Bergen. 1998. Efficient Collision Detection of Complex Deformable Models Using AABB Trees. *J. Graph. Tools* 2, 4 (jan 1998), 1–13. https://doi.org/10.1080/10867651.1997.10487480

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

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* (Hong Kong) *(VRST '10)*. Association for Computing

Machinery, New York, NY, USA, 63–70. https://doi.org/10.1145/1889863.1889874
Muiris Woulfe and Michael Manzke. 2009. A Framework for Benchmarking Interactive Collision Detection. In *Proceedings of the 25th Spring Conference on Computer Graphics* (Budmerice, Slovakia) *(SCCG '09)*. Association for Computing Machinery, New York, NY, USA, 205–212. https://doi.org/10.1145/1980462.1980501

A APPENDIX: POWERSHELL SCRIPT

A.1 Get IP Address of VM

```
get-vm -Name VMName | select -ExpandProperty
    networkadapters | select ipaddresses
```

A.2 Turning On/Off VM

```
param(
    [string] $vmName,
    [string] $op
)

$checkpoint = 'uploadTest'

if($op -eq 'start'){
    Restore-VMSnapshot -VMName $vmName -Name
    $checkpoint -Confirm:$false
    Start-VM -Name $vmName
    while ((get-vm -Name $vmName).state -ne '
    Running') {
        Write-Output "Waiting for $VMName to run"
        start-sleep -s 5
    }
} elseif ($op -eq 'stop') {
    Stop-VM -Name $vmName
}
```

A.3 Disconnect GPU From Host

```
Dismount-VMHostAssignableDevice -force -
    LocationPath [LocationPath]
```

A.4 Assign GPU to VM

```
Add-VMAssignableDevice -LocationPath [locationPath
    ] -VMName [VMName]
```

2022 IEEE-RAS 21th International Conference on Humanoid Robots (Humanoids)
November 28-30, 2022. Ginowan, Japan.

NaivPhys4RP - Towards Human-like Robot Perception
"Physical Reasoning based on Embodied Probabilistic Simulation"

Franklin Kenghagho K.[1], Michael Neumann[1], Patrick Mania[1], Toni Tan[2],
Feroz Siddiky A.[1], René Weller[2], Gabriel Zachmann[2] and Michael Beetz[1]

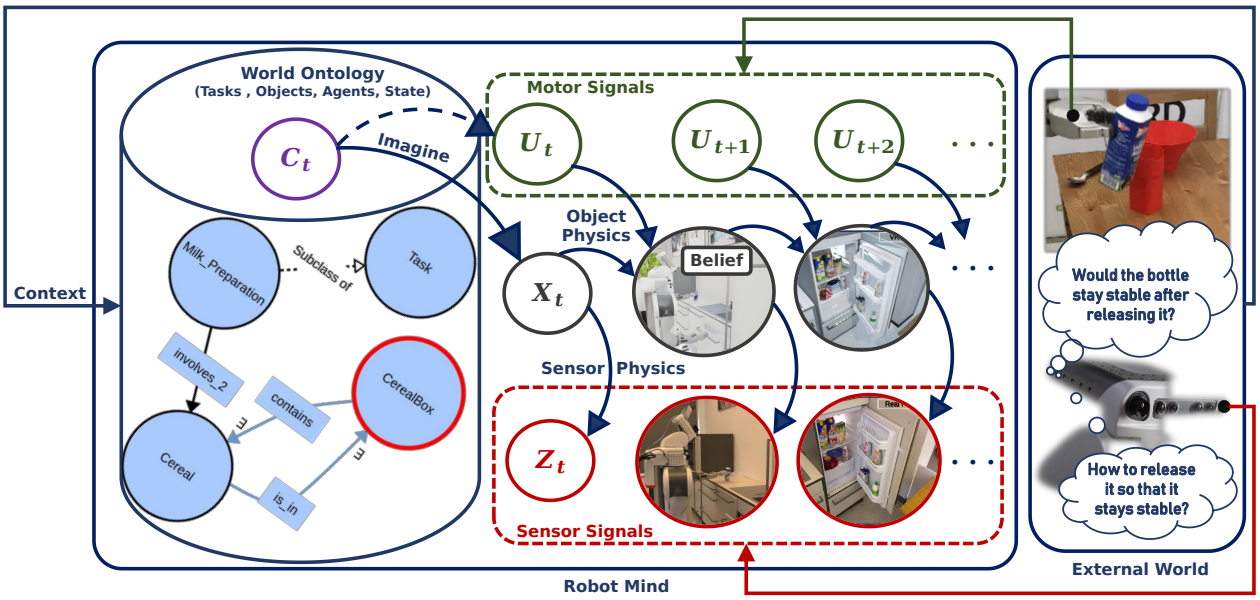


Fig. 1: Beyond static scenes, sensory information, and what-, where-questions. Commonsense and especially intuitive physics, also coined as dark matter of perception, is a key for perception in dynamic and human-centered scenes. Perception as inner realistic world construction that anticipates and explains the world state as well as observations in an explainable manner, with reasonable computational resources. We propose a white-box and causal generative model of perception in this paper.

***Abstract*— Perception in complex environments especially dynamic and human-centered ones goes beyond classical tasks such as classification usually known as the what- and where-object-questions from sensor data, and poses at least three challenges that are missed by most and not properly addressed by some actual robot perception systems. Note that sensors are extrinsically (e.g., clutter, embodiedness-due noise, delayed processing) and intrinsically (e.g., depth of transparent objects) very limited, resulting in a lack of or high-entropy data, that can only be difficultly compressed during learning, difficultly explained or intensively processed during interpretation. (a) Therefore, the perception system should rather reason about the causes that produce such effects (how/why-happen-questions). (b) It should reason about the consequences (effects) of agent-object and object-object interactions in order to anticipate (what-happen-questions) the (e.g., undesired) world state and then enable successful action on time. (c) Finally, it should explain its outputs for safety (meta why/how-happen-questions). This paper introduces a novel white-box and causal generative model of robot perception (NaivPhys4RP) that emulates human perception by capturing the Big Five aspects (FPCIU)[1] of human commonsense, recently established, that invisibly (dark) drive our observational data and allow us to overcome the above problems. However, NaivPhys4RP particularly focuses on the aspect of physics, which ultimately and constructively determines the world state.**

*This work was not supported by any organization

[1] with Institute for Artificial Intelligence, Mathematics and Computer Science, University of Bremen, Germany fkenghag@uni-bremen.de

[2]is with Computer Graphics Research Group, Mathematics and Computer Science, University of Bremen, Germany toni@uni-bremen.de

I. INTRODUCTION

Manipulation/action in human-centered environments requires perception systems to inform about the state of the world. However, the actual perception systems are struggling against the extreme dynamicity of such environments as well as the safety required. On the one hand, **(a) sensor information are very limited**. With extrinsic and intrinsic limitations such as occlusion, delayed processing, missing or poor depth for smooth and glass objects, attempts to solely rely on these sensory information lead to a situation where compression while learning, interpretation and processing speed are no more efficient due to lack of or higher entropy in the data (e.g., hard pose estimation) [1]. On the other

[1]Functionality, Physics, Causality, Intention, Utility

hand, **(b) these systems can only difficultly anticipate (undesired) states of the environment** given (a). Imagine the robot holding a plate containing a bowl and trying to open the drawer such as depicted by Figure 2.1, despite the fact that the robot camera is focused on the drawer, it should still be aware of the state of the bowl. Another scenario is the case of a robot trying to pour some milk from a bottle into a mug (see Figure 2.3).

Fig. 2: Physical reasoning for perception in dynamic scenes

Notice that success depends on the robot's understanding of the milk's fluid dynamics and how to control it by manipulating the bottle in order to ensure that the milk will neither fall out of the mug, the mug will not spill, nor the mug will be overfilled (Frame 3). On frame 2, the robot should ensure that the blue milk will not fall after releasing it, which desired state does not only depend on the table's physical relief but also on some bottle's physical parameters such as the shape, volume, mass, content and height [10]. Visual servoing has been an attempt to catch this scene dynamicity, however it is not only just reactive rather than anticipative but not robust to sensory limitations mentioned above. Finally, **(c) robotics in human-centered environments should also ensure safety and a step towards this goal is making the robots understand what they are perceiving and doing, in order words our models should not only be explainable but explainable based on causality rather than associativity** unlike most recent developments on explainability [10]. Though Deep Learning (DL) has shown great prowess on some perceptual classification tasks, there are more and more evidence that simply trying to compress huge amount of data, especially when the data entropy becomes high, fail to catch understanding. Slight modifications of only few pixels in images cause radical changes in results or a DL-based model telling that a train has been detected in the plate [1]. Given these issues, we ask ourselves how biological agents, at least humans, overcome them. In this regard, there are at least two observations. Firstly, (1) Physics constructively and ultimately determines the world state. Secondly, (2) there are more and more evidences, in contrast to David Marr's view of perception, that perception mostly goes from the inside out, where a mental intuitive physics engine continuously generates, simulates and maintains models of the world, which are then updated using sensory information [10, 8, 4]. Such a perception theory is illustrated by Figure 1.

In this paper, we contribute in addressing the three issues mentioned above (a-c) by:

- proposing a **complete, practical, and modular** architecture of perception systems, coined as **NaivPhys4RP (Naive Physics for Robot Perception)**, that leverages the physics that manipulated scene objects as well as the agent's sensory organs undergo to anticipate and explain the state and observation of realistic worlds in an explainable manner with reasonable computational resources.
- providing a **proof of concept for NaivPhys4RP** by demonstrating it on different challenging scenarios, namely object-related (transparency, occlusion), task-related (i.e., pose estimation, stability check) and domain-related (kitchen, medical laboratory).
- Showing that **NaivPhys4RP substantially considers the Big Fives requirements FPCIU (Functionality, Physics, Causality, Intent, Utility)**[10] for achieving human-level perception recently established.

II. RELATED WORK

Despite the increasingly intensive research on how biological agents, at least humans, do intuitively grasp the physical laws governing the state of the physical world around them from limited sensory information and how they apply such knowledge, commonly referred in the literature to as commonsense, intuitive, naive or folks physics, to anticipate the state or interpret observations, the results remain on the one hand abstract (e.g., higher-level hypotheses/findings) from the Psychology community [10] and primitive (e.g., 2D-, simplistic and unrealistic worlds, partial theories (e.g., disembodiedness)) from the community for computational sciences on the other hand [3] . This being said, we will mostly focus on the core computational theories underlying these research works as well as the two observations (1-2).

Embodied Simulation. Based on evidences, (Hesslow, 2002) [4] constructed a theory of conscious thought as embodied mental simulation, where the brain can simulate an action in an overt manner (i.e., without realization in real world) and simulate the perception of that action's effects usually referred to as Mind Eye, Ear, etc. Depending on the action's effects, the agent might decide to simulate the action in a covert manner, where the action is actually performed in the physical world. That action's effects are then perceived through the physical sensor organ (e.g., eye) and the cycle restarts. Note that, it is also possible to start the loop with a simulated perception from the mind eye (i.e., imagination). It is argued that the theory provides a way to the supportive interactions between motor, sensory, cognitive functions and the internal representations of the world, a way to anticipation a.k.a. prospection and emphasizes the essence of anticipation in cognition. (Cassimatis et al., 2004) emphasizes the advantages of the simulation theory of cognition and show how it constitutes a potential solution to many problems encountered in robotics.

Intuitive Physics. There have been more and more evidences that human cognition, yet at earlier months of life, can understand the physics governing the behavior of objects in the physical world and then use this knowledge to anticipate physical changes (i.e., object fall, object pose), which then enables successful and smooth action in realtime. Notice that this happens without prior education in physics or

knowledge of the physical parameters of the world such as mass, friction, which are not only intractable and inexplicable for uneducated people in Physics but would not explain the smoothness and realtimeness of actions. In this regard, most research works have been supporting the hypothesis of a common physics engine that roughly infers the physical parameters (e.g., friction, mass) of the world from sensory information and then uses them as inputs to a forward simulation through the engine in order to anticipate events and states. Moreover, it has been shown that deviations in common physical reasoning could go back at least to the extrinsic (e.g., inaccurate physical parameters) and intrinsic (e.g., unobservable parameters) uncertainty of the physical phenomena, which parameters could be refined over time for more accurate reasoning. Researchers, especially Joshua Tenenbaum and his colleagues have considerably argued on how intuitive physics is essential for perception from limited sensory information (e.g., observing a car moving, and after it passes behind an occluding wall, we can still predict when it will appear at the other extremity of the wall) and have termed it as dark matter of perception in the sense that it is not directly graspable from sensory information but significantly contributes in generating these information [10]. However, Davis and his colleagues objected to the simulation theory for intuitive physics, claiming on the one hand the intractable computational resources required and on the other hand the failure of the simulation theory to the conjunction-fallacy effect. Recently, (Bass et al., 2022)[2] replied to Davis's objection with a theory of partial simulation. In sum, these works on intuitive physics stresses the physical, probabilistic, partial and emergent nature of the simulation theory of Hesslow.

Perception as Controlled Hallucination. (Anil Seth, 2018) [8] argues on the limitations of sensory information and flaws in David Marr's standard theory of perception (i.e., bottom-up information processing) and regarding this issue he elaborated a theory of perception based on evidences, where the brain, so-called bayesian, continuously generates, simulates expectations of the world state (i.e., hallucinations) and updates this expectations with the few available sensory information (i.e., control). This dominant top-down view of perception was already argued by (Ralf Moeller, 1996), defining perception as the process of anticipating sensory consequences of actions .

Imagination-capable Belief State (ICBS). Finally, we (Mania et al., 2021) [6] recently proposed a very rich inner representation of the world, also known as semantic digital twin as it aims at replicating the real world in photo-realistic and physics-faithful virtual environments (i.e., game engines) grounded in the world ontology for more semantics. Then, we showed how such a representation could be used to validate and refine the outputs of a traditional perception system. In this paper, we continue this work with regard of the above theories by enlarging the capabilities of these mental world representations to **embodied probabilistic** simulations and provide an architecture of perception systems, intrinsically based on such simulations and other aspects of commonsense such as the process context, that can perform physical reasoning to cope with the problems (a-c).

III. Architecture

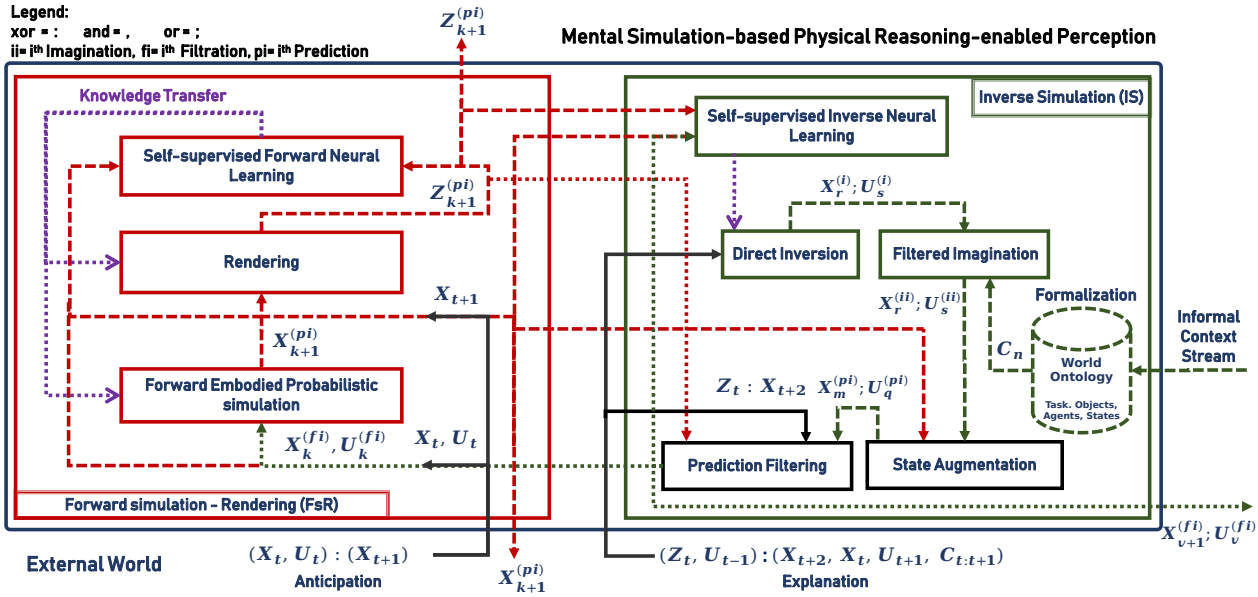


Fig. 3: The robot observations Z_t and actions U_{t-1} are tightly coupled through a sufficiently rich inner model of the world X_t that allows through a forward simulation and rendering module (FsR) to anticipate the world state X_{t+1} ($X^{(pi)}_{t+1}$) and its observations Z_{t+1} ($Z^{(pi)}_{t+1}$), then to explain the world observation Z_{t+1} ($X^{(fi)}_{t+1}$) and its state X_{t+1} ($X^{(fi)}_t$, $U^{(fi)}_t$) through an inverse simulation module (IS). X_t emerges overtime through a complementary and white interaction loop between IS and FsR, where IS constructively infers the causes whose consequences through FsR match the observed or intended consequences.

A. Problem formalization

In regard to the above theories, we formalize the problem addressed by NaivPhys4RP in four steps. (i) We model the world state, as shown by Figure 1, as a **S**ituated (i.e., take place in a context) **P**artially-**O**bservable (i.e., only partial sensor data) **H**idden (i.e., not directly accessible information) **M**arkov **P**rocess (i.e., state dependency) (SPOHMP) that evolves through the physics that scene entities (e.g., objects, robots, sensors) undergo. (ii) We model the hidden state a.k.a. belief of the SPOHMP as ICBS described earlier. (iii) Then, we regard perception as taskable through queries [10, 5] and these perceptual queries are clustered into anticipatory (i.e., consequences given causes) and explanatory queries (i.e., causes given consequences), that are abstracted as bayesian/markovian inference tasks. However, note that an actual accurate and rich belief of the world state is the informational source for answering these questions. Such a belief is continuously filtered over time through emulation of the SPOHMP. (iv) Finally, we efficiently implement the four main operators of the rao-blackwellized particle filter [7], however modified to five operators, which is a generic, practical and constructive approach to simultaneously emulate the SPOHMP and address the bayesian inference tasks just mentioned (markov-blanketed), through embodied, physics-faithful, photo-realistic, probabilistic, partial and ontology-grounded simulations. This formalization is summarized by the equations (1) below:

$$\begin{cases} \boldsymbol{X}_t^* \sim \boldsymbol{P}(\boldsymbol{X}_t|\boldsymbol{U}_{0:t-1},\boldsymbol{Z}_{0:t},\boldsymbol{C}_{0:t}) & \text{, actual belief} \\ \boldsymbol{X}_{t+1}^* \sim \boldsymbol{P}(\boldsymbol{X}_{t+1}|\boldsymbol{U}_t,\boldsymbol{X}_t,[\boldsymbol{C}_{t+1}]) & \text{, state anticipation} \\ \boldsymbol{X}_{t+1}^*,\boldsymbol{U}_t^* \sim \boldsymbol{P}(\boldsymbol{X}_{t+1},\boldsymbol{U}_t|\boldsymbol{U}_{t+1},\boldsymbol{C}_{t:t+1},\boldsymbol{X}_t,\boldsymbol{X}_{t+2}) & \text{, state explanation} \\ \boldsymbol{Z}_{t+1}^* \sim \boldsymbol{P}(\boldsymbol{Z}_{t+1}|\boldsymbol{X}_{t+1}) & \text{, observation anticipation} \\ \boldsymbol{X}_{t+1}^* \sim \boldsymbol{P}(\boldsymbol{X}_{t+1}|\boldsymbol{U}_t,\boldsymbol{X}_t,\boldsymbol{Z}_{t+1},\boldsymbol{C}_{t+1}) & \text{, observation explanation} \end{cases} \tag{1}$$

- X, is the world's hidden state (e.g., a digital twin)
- Z, is the object/world observation (e.g., rgbd images)
- U, is the motion control (e.g., joint values, forces)
- C, is the process context (e.g., object + task knowledge)

Following are the five main operators of the modified rao-blackwellized particle filter ($mRBFP$):

- Belief ***initialization***, $X_0^{(i)} \sim P(X_0|C_0)$
  amortized initialization, $X_0^{(i)} \sim P(X_0|C_0,Z_0)$
- Belief **prediction**, $\tilde{X}_{t+1}^{(i)} \sim P(\tilde{X}_{t+1}|X_t,U_t)$
- Belief ***augmentation***, $X_{t+1}^{(i)} \sim P(X_{t+1}|\tilde{X}_{t+1},C_{t+1})$
  amortized augmentation, $X_{t+1}^{(i)} \sim P(X_{t+1}|\tilde{X}_{t+1},C_{t+1},Z_{t+1})$
- Belief **weighting**, $W_{t+1}^{(i)} \approx P(Z_{t+1}|X_{t+1})$
- Belief **filtering**, $X_{t+1}^{(i)} \sim \frac{W_{t+1}^{(i)}}{\sum W}$

Note that i, t, $[.]$ and $\sim$ respectively denote the particle index, the time index, optional priors and the argmax probabilistic sampling. Though the variable U is not sampled by the above operators of a mRBPF, we show how the third equation in (1) can be solved using the general principles of these operators. Finally, the architecture on Figure 3 essentially computes these operators to solve the inference tasks in (1).

B. Ontology-Grounded Physico-Realistic Belief State (X_t)

An Imagination-Capable Belief State (ICBS) goes beyond usual semantic scene graphs (objects' description and relations among objects) and incorporates the scene geometry (e.g., articulated 3D models), scene physics (e.g., gravity, friction, mass, forces, viscosity, waves), scene agents (e.g., operating robots' motorics and sensorics), scene ontology (i.e., semantics). The ontology is a formal description of fundamental and common truths about task-, agent-, object and state-related concepts, their properties and relationships among them in the scene. Depending on the particular scene under study, the ontology can be enriched with typical knowledge. It is also worth noting that state-related concepts that are unusual in most ontology definitions model in NaivPhys4RP a higher-level semantics of the effects of physics (e.g., through action) on the world. For a possibly lossless representation and reliable simulation of the belief, the latter is directly represented in a photo-realistic and physics-faithful game engine, grounded in a rich scene ontology, and interfaces are provided to assert, modify, simulate and query it.

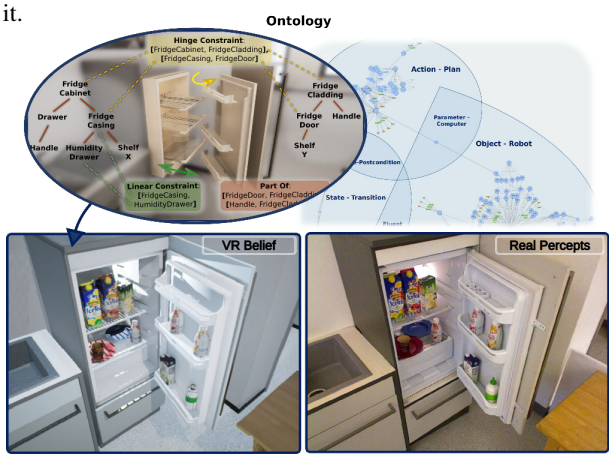


Fig. 4: Belief (left), real world (right), world ontology (top).

C. Forward Simulation - Rendering (FsR)

1) Anticipation: This FsR module, as reported in Figure 3's caption, is mainly responsible for anticipating the observations $Z_{t+1}^{(pi)}$ and the states $X_{t+1}^{(pi)}$ as consequences of the causes $X_t^{(fi)}$ and $U_t^{(fi)}$. Note that the superscripts pi and fi respectively denote the prediction p and the filtering f of particle i. Given our realistic mental simulations, these inference tasks are performed straight-forward as shown by the resolution equations (2) below:

$$\begin{cases} \boldsymbol{X}_{t+1}^{(pi)} \approx \boldsymbol{Simulation}_{\lambda_s}(\boldsymbol{X}_t^{(fi)},\boldsymbol{U}_t^{(fi)}) & \text{, state} \\ \boldsymbol{Z}_{t+1}^{(pi)} \approx \boldsymbol{Rendering}_{\lambda_r}(\boldsymbol{X}_{t+1}^{(pi)}) & \text{, observation} \end{cases} \tag{2}$$

The accuracy of these operations in (2) lies in the parameters λ_s and λ_r and we achieve it in two steps: targeting of realisticness (section III-C.2) and integration of uncertainty about physics (section III-C.3). For achieving a reasonable time complexity for the set of particles during inference, we rely as described below, on many cues such as parallelism,

neural accelerators, Rao-Blackwellization and Partiality (section III-C.4).

2) Embodied Realistic Simulation: In the project **RobCog** (Robot Cognition: `robcog.org`) , as illustrated by Figure 4 and 5, we demonstrated how a photo-realistic and physics-faithful virtualization of everyday manipulation scenes (e.g., kitchens, medical labs) in the game engine Unreal Engine (UE) can be achieved, grounded in a large scene ontology (KnowRob-SOMA: `knowrob.org`) and used to perform human demonstrations of manipulation activities through a realistic human avatar so that rich datasets (NEEMs: Narrative-Enabled Episodic Memories) are automatically collected for machine learning purposes. The project **DAO** (Deep Action Observer)[2] extends RobCog by observing humans in activity and projecting their actions and motions onto programmable human avatars in the virtual world (see Figure 5). The project **URoboSim** (Unreal Robot Simulator: `embodied-ai.org/papers/URoboSim.pdf`), as illustrated by Figure 1 and 5, extends RobCog by developing virtual robot agents with sensing capabilities that can mirror what a real robot is doing or demonstrate what the real robot will be doing.

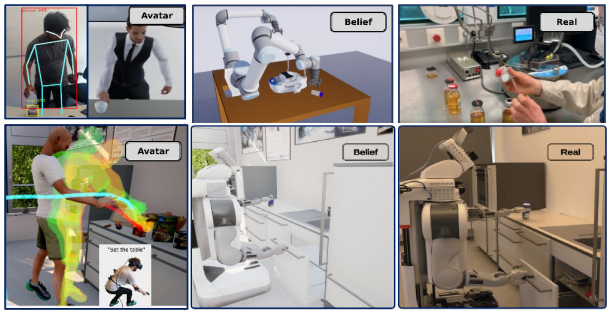


Fig. 5: RobCog (bottom-left), DAO (top-left), URoboSIM (real world in right and belief in left).

3) Uncertain Physics: Despite our ambition to target a realistic robot belief in appearance and physics, a perfect simulation remains challenging due to uncertainty about physical parameters like friction, mass, or object position in the world. In the belief X_t, uncertainty is partially considered in mRBPF as many belief particles are simulated, weighted, and then sampled based on their weights. However, this could require many belief particles to reach the right physical parameters, especially for continuous physical quantities. Collision and forces are fundamental in estimating the physical dynamics of objects in simulations. Therefore, to reduce the number of particles needed, we propose embedding uncertainty directly into object geometry, precisely, the underlying acceleration data structure. Within the scope of this paper, we applied the idea on top of *Inner Sphere Tree (IST)* [9]; nevertheless, it applies to other algorithms as well. As an example (see Figure 6), imagine the robot in Figure 5 trying to throw a blue milk bottle in the dustbin. In this case, the input is no more a single mass value of the object before its free-fall but rather a probabilistic distribution of its mass, friction, or object position. Likewise, the output will be a probabilistic distribution of its location when it finishes the fall. This approach considerably reduces the number of belief particles representing such distribution.

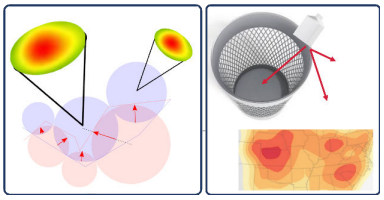


Fig. 6: (left) Elementary forces during a single simulation step between thrown bottle (blue) and dustbin (red), and (right) probabilistic distribution of bottle's location after simulation.

4) Temporal Efficiency: In this section, the cues we rely on to accelerate FsR on the set of belief particles are presented. **(i)** Rao-blackwellisation: Uncertain physical simulation is regarded as an emulation of the analytical estimation of probabilistic distributions of some continuous variables in X_t, reducing the number of belief particles needed for emulating the SPOHMP. **(ii)** Parallel FsR : We demonstrated in a Master thesis how, thank to cloud computing, FsR could be parallelized over the set of belief particles as shown by Figure 7. **(iii)** Partial simulation: SPOHMP is intrinsically partially-observable and this is taken into account during the emulation as X_t only get sampled incrementally through the augmentation operator of mRBPF. **(iv)** Self-trained neural accelerators: In [5], we demonstrated how a perception system can efficiently train on auto-generated data (e.g., NEEMs) from embodied and situated simulation to infer advanced semantic graphs of the scene. Instead of proceeding through procedural operator of the game engine, neural operators (λ_s and λ_r) trained from NEEMs could be integrated in game engines or operate beside them, as shown by the violet arrows on Figure 3. **(v)** Prediction as straightforward simulation: Finally, this is another major advantage of our approach over traditional symbolic and qualitative approaches which do not only require a huge gymnastics to sample from multidimensional probabilistic distributions, but also sample states that are not physically plausible within a certain context.

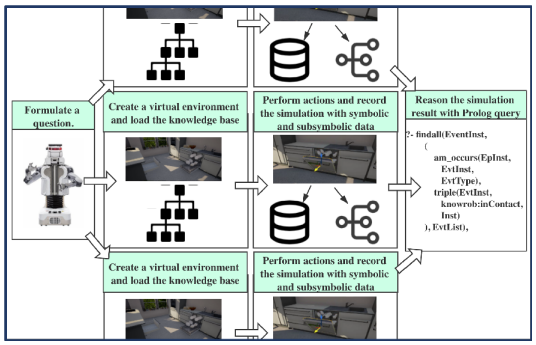


Fig. 7: Accelerating FsR through parallelism.

[2]dropbox.com/s/60fweieljn9pbky/deep-action- observer.pdf?dl=0

D. FsR-based Inverse Simulation (IS)

1) Explanation: This module is mainly responsible for processing explanatory questions such as presented in (1), in a constructive manner based on FsR, that makes it white and therefore interpretable and explainable, since FsR is eihter. Intuitively, the goal is to generate states $X_{t+1}^{(fi)}$ that explain observations Z_{t+1} and state-action couples $(X_t^{(fi)}, U_t^{(fi)})$ that explain desired states X_{t+1} and for achieving this, the remaining four main operators of mRBPF have to be computed.

2) (Amortized) Belief Initialization: It is intractable to merely sample these particles from the initial space of states. As humans rely on intuitive physics as a domain of commonsense to understand the physics that the world surrounding them undergoes, they do likely leverage commonsense about their operating scenes also referred to as context to formulate high-quality expectations about the scene state as far as the nature of objects and their natural (e.g., spatial) configurations are concerned in order to achieve estimation of the world state from limited sensory information. We model such a cognitive function in three core steps.

(i) Context formalization: As you can see from the architectural figures 3 and 1, the context that conceptually characterizes the scenes the robot operates in is either vaguely provided to the system under any communication modality such as text, audio, or even formally provided and directly stored within a shared memory. In the former case, The goal of the formalization step will then be to circumscribe a sufficiently rich field of concepts and relations among those concepts that underlie the target scene. Let assume that the most common input modality for context is textual, then our framework PRAC[3] (Probabilistic Action Cores) can be used to formalize such a vague specification, such as illustrated by Figure 8.

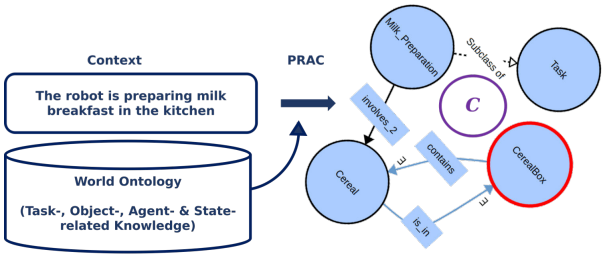


Fig. 8: Context formalization.

(ii) Context-specific imagination: Once the context has been formalized, possible states of the world can be imagined.

Fig. 9: Context-specific imagination of world state.

As shown by Figure 9, we demonstrated in [5] how situated and embodied datasets for perception systems could be generated from context-specific imagination. For preparing the breakfast, there is a need for cereal which is in the cereal box, a bowl and a spoon which can be and is usually inside the bowl.

(iii) Amortization: Despite the considerable reduction of the world state space through context-specific imagination, still there remains a bit of vagueness for instance in terms of number of objects and concrete spatial configurations. In order to amortize this combinatorial explosion, we employ a greedy direct (unconscious) perception approach of the scene, neurally trained on imagined datasets, to compress the state space. Then, the optimistic results of the neural learner are filtered based on the imagination (e.g., if knife detected then likely spoon because coffee drinking). We developed, RobotVQA (Robot Visual Question Answering) [5] for supporting the taskable and cognitive perception system RoboSherlock[4]. Notice that this step is realized by the *direct inversion* and *filtered imagination* modules on Figure 3.

3) (Amortized) Belief Augmentation: Notice that the belief initialization is only based on partial observations and the initialization is therefore only partial. Then, forward simulating from such a partial initialization is not enough to achieve convergence of belief particles towards the world state. For this reason, a belief augmentation is performed after each prediction $X_{t+1}^{(pi)}$ where identical operations as in the initialization step are used based on the actual observation Z_{t+1} and context C_{t+1}, and the results are then aggregated to the prediction for enriching it. At the belief initialization, there is no aggregation because the prediction is empty.

4) Belief Weighting: The weights of belief particles are $W_{t+1}^{(i)}$ computed by the straight-forward operation below:

$$\begin{cases} \boldsymbol{D}_{t+1}^{(i)} \approx \boldsymbol{Distance}_{\lambda_d}(\boldsymbol{Z}_{t+1}^{(pi)}, \boldsymbol{Z}_{t+1}) & \text{, actual} \\ \boldsymbol{W}_{t+1}^{(i)} \approx \boldsymbol{D}_{t+1}^{(i)} + \boldsymbol{W}_{t}^{(i)} & \text{, cumulative} \end{cases} \tag{3}$$

Intuitively, $D_{t+1}^{(i)}$ measures how close to the real partial observation Z_{t+1} the observation $Z_{t+1}^{(pi)}$ of the realistic rendering of the predicted belief $X_{t+1}^{(pi)}$ is (see Figure 4). For all the observations up to $t+1$ (i.e., total observations), the cumulative distance is expressed by $W_{t+1}^{(i)}$.

5) Belief Filtering: Finally, the belief particle are filtered through a random sampling with replacement according to their weights from the set of belief particles: $X_{t+1}^{(i)} \sim \frac{W_{t+1}^{(i)}}{\sum W}$. This ensures the convergences of the belief towards the real world state.

6) State Explanation: We highlighted earlier in this section that though the native main operators of a mRBPF do not support the explanation of states described as $\boldsymbol{X}_{t+1}^{*}, \boldsymbol{U}_t^{*} \sim \boldsymbol{P}(\boldsymbol{X}_{t+1}, \boldsymbol{U}_t | \boldsymbol{U}_{t+1}, \boldsymbol{C}_{t:t+1}, \boldsymbol{X}_t, \boldsymbol{X}_{t+2})$, their general principles can be employed to address the problem. Literally, given

[3] http://www.actioncores.org/

[4] http://robosherlock.org/, https://github.com/robosherlock

the actual belief X_t, we are looking for an action U_t^* within a context C_t that would transform X_t into a state X_{t+1} within a context C_{t+1} so that by applying the action U_{t+1} one could reach the target state X_{t+2} (e.g., how should I hold the milk bottle so that if I release it on the table, it will not fall). Notice foremost that this problem can be approximately broken into three problems according to rao-blackwellization namely (p1) $\boldsymbol{U_t^{(k)} \sim P(U_t|C_t)}$, (p2) $\boldsymbol{X_{t+1}^{(k)} \sim P(X_{t+1}|U_t, X_t, C_{t+1})}$ and (p3) $\boldsymbol{W^{(k)} \approx P(X_{t+2}|U_{t+1}, X_{t+1}, C_{t+1})}$. While (p2) and (p3) have already been solved by the FsR and Distance functions above, (p1) can be solved by sampling U_t according to the context C_t and the whole problem by filtering the U_t^* based on how good they turn X_t into the desired X_{t+2}. Notice the steps of a mRBPF except that U is the target instead of X. And since this work is about physical reasoning based on mental embodied simulations for perception, addressing the estimation of U to know about the state, does not only considerable goes beyond state estimation (e.g., action & motion planning required, U as joint states is not meaningful), but also emphasizes how perception, motorics and cognitive functions are strongly intertwined. In order to sample meaningful control commands U, we rely on CRAM (Cognitive Robot Abstract Machine)[5], an established cognitive architecture, that samples U from a bag of generic primitive action plans (see Figure 10), then contextualize it using the world ontology C and the world state X to finally produce joint states that can be directly realized by the virtual robots.

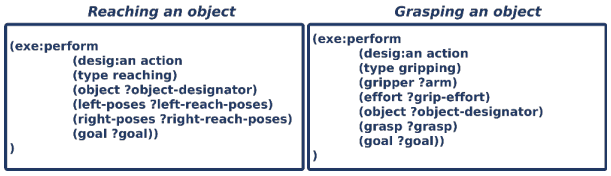


Fig. 10: Underspecified primitive action plans.

7) Temporal Efficiency: We leverage the following cues in order to achieve a reasonable time complexity for IS. **(i)** FsR's efficiency: IS is either a constructive approach based on FsR. **(ii)** Amortization: The use of self-trained neural accelerators for reducing the number of belief particles has been presented. **(iii)** Faster filtering: the belief particles are filtered based on a straight-forward computation of their importance weights. **(iv)** Faster convergence: The belief particles tend to converge quickly to the real world state since only few imaginary states are physically plausible before and after simulating.

IV. NaivPhys4RP and The Big Fives FPCIU

In a recent journal article [10], Tenenbaum and his colleagues identified five core aspects (FPCIU) of human commonsense, hierarchically organized, namely **F**unctionality, **P**hysics, **I**ntent, **C**ausality, and **U**tility to consider in order to hope human-level perception in Artificial Intelligence.
(i) Causality: As the basis for understanding, it is characterized by the elicitation of cause-effect relationships for the sake of explaining and anticipating phenomena. On the one hand, NaivPhys4RP inherently relies on physical simulation which itself relies on the integration of physical causality (i.e., laws of physics). Beyond physical causality, the context C encodes other forms of causality such as the functional causality (e.g., Milk preparation causes usage of certain products).
(ii) Physics: NaivPhys4RP obviously achieve commonsense physics through its ability to track the physical causality.
(iii) Functionality: Most objects in human-centered environments are functional and these functions are very decisive in experiencing (e.g., categorizing) the world around us though not directly observable from sensory information. NaivPhys4RP achieves this through functional causality.
(iv) Intent: in NaivPhys4RP, the agents's actions are modeled by the layer U even if in the current formalization, only the actions of the operating agent are explicitly represented. DAO can help in tracking and integrating other agents' actions. Moreover, the layer C (see Figure 8) partially captures the agents' intentions however can be made more explicit with an intent layer on top of U.
(v) Utility: humans act rationally by making choices that maximize their utility function (e.g., survival, travel cost, operation duration, success). NEEMs collected from NaivPhys4RP can be used for the learning of the agent's preferences such as scene objects, their spatial dispositions, the agent poses for grasping and perceiving different objects.

V. Experimentation

As a proof of concept, we demonstrate NaivPhys4RP in the following few challenging scenarios. We provide more information about the experiments in the demo video attached to this paper.
(i) 6D-Pose of In-hand Objects: Robots are usually unaware of the pose of objects they hold, which is critical for meaningful manipulation.

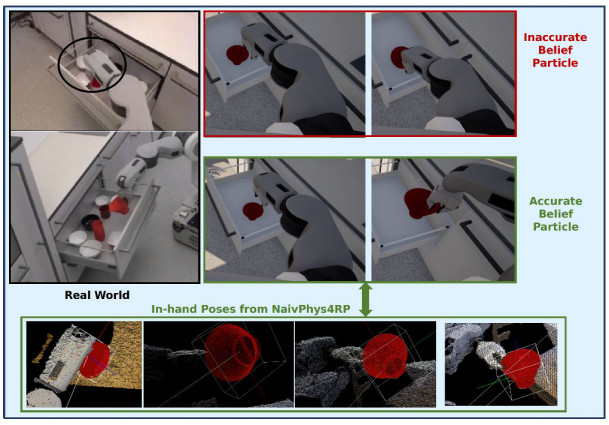


Fig. 11: NaivPhys4RP estimates in-hand poses.

(ii) 6D-Pose of Transparent & Smooth Objects: Certain objects exhibit a poor depth from optical depth cameras due to absorption, retransmission or specular rather than scattered reflection of emitted light rays. Figure 12 illustrates how

[5]http://www.cram-system.org/

NaivPhys4RP overcome the issue and estimate the pose of such objects.

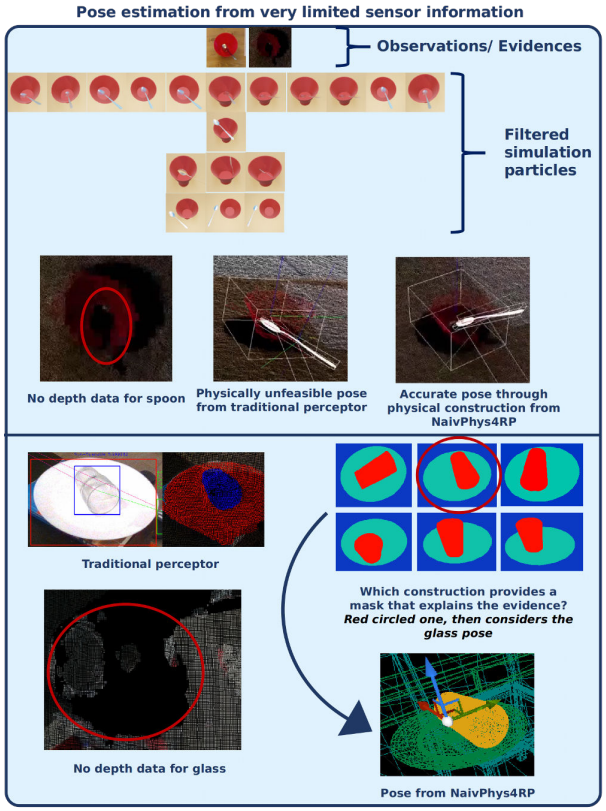


Fig. 12: NaivPhys4RP estimates poses from poor depth.

(iii) Object's Semantic Stability: How to place the milk bottle so that it does not fall?

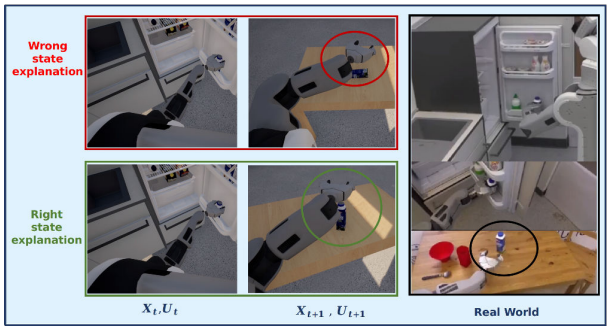


Fig. 13: NaivPhys4RP explains future desired state of world.

(iv) Generalizability: TraceBot:

Finally, we demonstrated how the approach is generalizable and can be applied to more complex, especially mission-critical applications such as TraceBot, a project that robotizes the process of medical sterility testing. Figure 14 shows how NaivPhys4RP can localize subtle tool parts and mirror the robot failures (`www.tracebot.eu`).

VI. CONCLUSIONS

We proposed in this paper a practical framework NaivPhys4RP with a proof of concept for scaling robot perception towards complex environments such as dynamic and human-centered scenes (i.e., motion, limited sensory information, safety). To emulate human perception, NaivPhys4RP es-

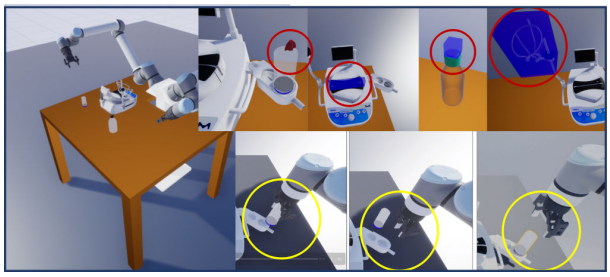


Fig. 14: NaivPhys4RP in TraceBot.

sentially relies on realistic, embodied, physics-faithful and partial simulations grounded in the world ontology. In doing this, NaivPhys4RP substantially leverages commonsense knowledge about the world and foremost intuitive physics. In the future, we aim at a stable implementation of NaivPhys4RP with a focus on integrating the core components, but also on a systematic and quantitative evaluation and finally on an explicit integration of the FPCIU such as described in section IV.

ACKNOWLEDGMENT

This scientific work is partially funded by the projects DFG EASE CRC 1320, EU TraceBot (grant agreement No 101017089), and BMBF ILIAS (grant no. 01DR19001B).

REFERENCES

[1] Pieter Adriaans. "Learning as Data Compression". In: 2007.

[2] Ilona Bass et al. "Partial mental simulation explains fallacies in physical reasoning". In: (2022).

[3] Jiafei Duan et al. *A Survey on Machine Learning Approaches for Modelling Intuitive Physics*. 2022.

[4] Germund Hesslow. "Hesslow, G. Conscious thought as simulation of behaviour and perception." In: *Trends in cognitive sciences* (2002).

[5] Franklin Kenghagho Kenfack et al. "RobotVQA — A Scene-Graph- and Deep-Learning-based Visual Question Answering System for Robot Manipulation". In: 2020.

[6] Patrick Mania et al. "Imagination-Enabled Robot Perception". In: 2021.

[7] Kevin Murphy and Stuart Russell. "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks". In: 2001.

[8] Anil K. Seth. "Consciousness: The last 50 years (and the next)". In: *Brain and Neuroscience Advances* (2018).

[9] Rene Weller and Gabriel Zachmann. "Inner sphere trees for proximity and penetration queries." In: 2009.

[10] Yixin Zhu et al. "Dark, Beyond Deep: A Paradigm Shift to Cognitive AI with Humanlike Common Sense". In: *Engineering* (2020).