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].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Web3D '22, November 2–4, 2022, Evry-Courcouronnes, France
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
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.

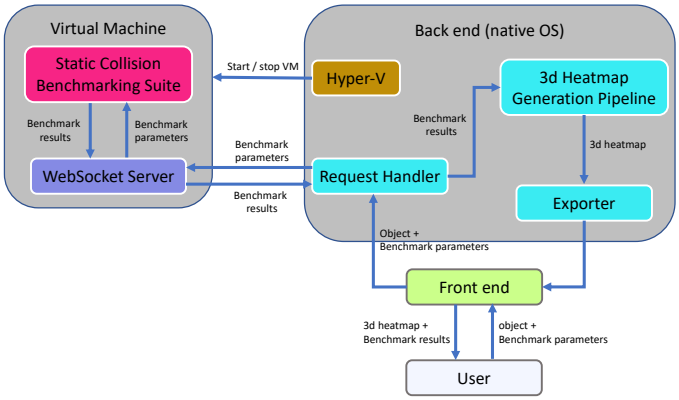

Figure 1: System Overview of our proposed framework, which extends the capability of openCollBench to benchmark user-uploaded algorithms in a secure virtualization environment

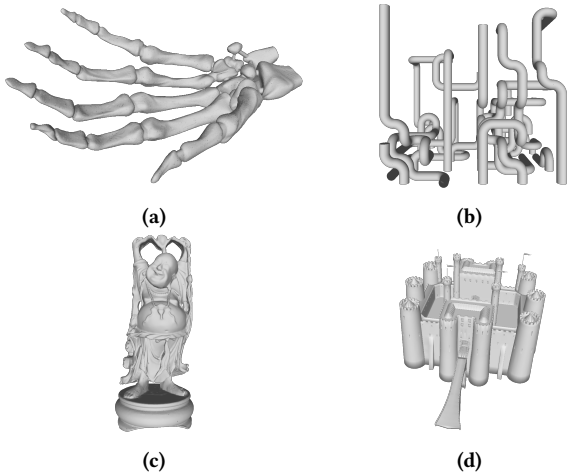

Figure 2: The objects we used in our timings: (a) hand, (b) pipes, (c) happy buddha, and (d) castle.

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

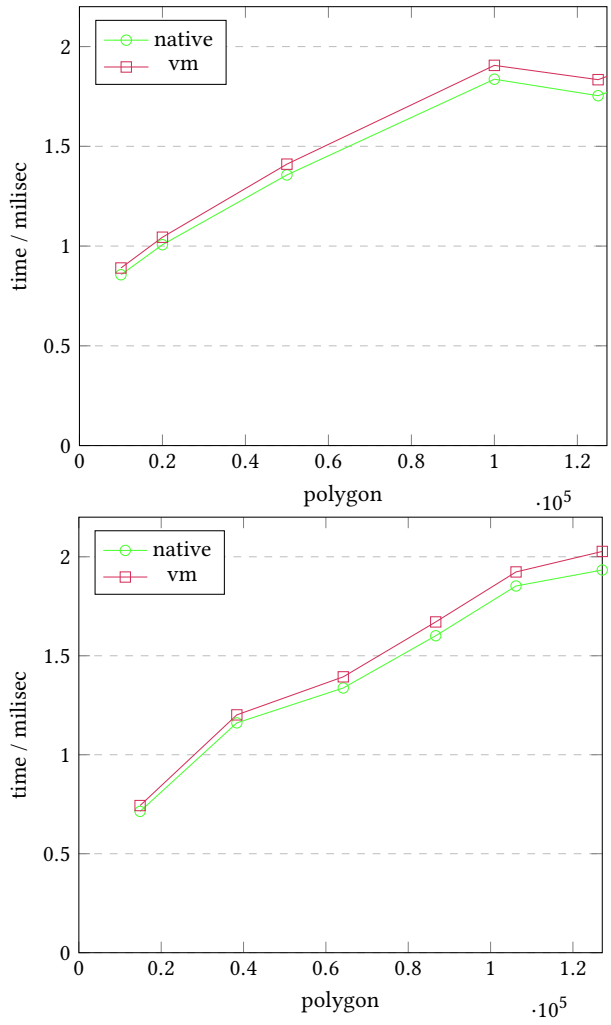

Figure 3: Average collision query time for the object Castle and happy buddha in native and virtualization environment. The delta are very similar for all objects.

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>.

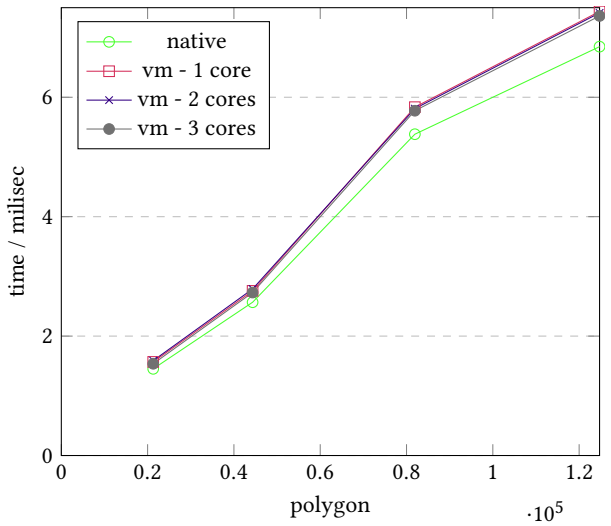

Figure 4: Average collision query time for the object Pipes in native and virtualization environment using 1 core, 2 cores, and 3 cores.

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]
```