2023 IEEE International Conference on Edge Computing and Communications (EDGE) | 979-8-3503-0483-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/EDGE60047.2023.00049

A Comprehensive Performance Evaluation of Procedural Geometry Workloads on Resource-Constrained Devices

Edon Govori, Ilir Murturi, and Schahram Dustdar

Distributed Systems Group, TU Wien, Vienna, Austria.

***Abstract*—In recent years, visualizing high-quality 3D content within modern applications (e.g., Augmented or Virtual Reality) is increasingly being generated procedurally rather than explicitly. This manifests in producing highly detailed geometries entailing resource-intensive computational workloads (i.e., Procedural Geometry Workloads) with particular characteristics. Typically, workloads with resource-intensive demands are executed in environments with powerful resources (i.e., the cloud). However, the enormous amount of data transmission, heterogeneous devices, and networks involved impact overall latency and quality in user-facing applications. To tackle these challenges, computing entities (i.e., edge devices) located near end-users can be utilized to generate 3D content. Our objective within this paper is to evaluate performance and power consumption when executing procedural geometric workloads on resource-constrained edge devices. Through extensive experiments, we aim to comprehend the limitations of different edge devices when generating 3D content under different configurations.**

***Index Terms*—Edge Architectures, Computational Workloads, Computing Continuum, Resource-Constrained.**

I. Introduction

Applications such as those within Augmented or Virtual Reality (AR/VR) have unique software architecture requirements that must be able to handle latency, data transmission, and performance. Visualizing and representing high-quality 3D digital models are computationally intensive, requiring significant data transfer to clients. While these applications are susceptible to latency delay, processing data on cloud servers involves high latency caused by data transmission over multiple networks, and heterogeneous devices [1]. In order to tackle these challenges, recent advancements have resulted in applications that can operate across all three computing tiers, including the edge, fog, and cloud computing tiers [2], [3].

Edge computing is a novel computing paradigm that allows computation to be performed very close to data sources. Migrating all computing tasks to the cloud has proven to be an effective method for processing data, as the cloud has a higher computing power concentration than devices located at the network edge. Despite this, the networks' bandwidth that transmits and receives data to and from the cloud hasn't grown substantially, leading to network bottlenecks as IoT devices generate more data [4], [5]. Edge resources, also known as edge devices, are often characterized as resource-constrained and heterogeneous regarding hardware and software capabilities. These devices can handle, store, and analyze data locally and provide fast and efficient services to end users. Furthermore, they can play a crucial role in supporting applications in several ways, e.g., executing computational tasks and processing data locally without transmitting it to the cloud.

Visualizing high-quality 3D model content has become a key requirement for various practical applications [6]. The accurate digital models are particularly useful in the context of simulating cities and are often used in maps for navigation, or the simulation of natural and social phenomena [7], [8]. Using 3D representations in near real-time applications often lacks in terms of quality and this is mainly impeded by the sheer volume of the 3D data that must be transferred between devices. To prevent the storage and transmission of a significant amount of data, we generate the geometric content procedurally instead of explicitly [9]. Procedural generation has been effectively utilized in numerous industries and applications where compelling and immersive virtual urban environments are crucial to enhance the visual experience [10], [11]. Generally speaking, procedural modeling can generate highly detailed geometry by repeatedly applying a set of rules that describe spatial transformations to basic shapes, referred to as an axiom (i.e., such as a box or quadrilateral). This approach starts with a simple primitive shape and generates more complex geometric structures. For instance, windows, or balconies on a wall can be created by dividing a rectangle into smaller rectangles and then applying extrusion operations and different textures to each of the smaller rectangles. Similarly, using additional rules the details on windows or doors can be produced by further dividing, extruding, and mapping textures to smaller shapes.

Procedural geometry is highly flexible in generating geometric data. This means that the representation of 3D environments in devices with minimal resources is possible under different configurations (i.e., by tuning the level of detail (LoD) [12] or margin). In addition, recent efforts show that faster procedural geometry generations can be achieved by executing workloads simultaneously in different devices [13]. Consequently, this enables procedural geometry workloads to be distributed on a wide range of devices in the computing continuum and improve overall performance. However, the computing continuum infrastructures are heterogenous environments; specifically, edge-based infrastructures consist of

2767-9918/23/$31.00 ©2023 IEEE
DOI 10.1109/EDGE60047.2023.00049

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

different devices featuring different hardware and software capabilities. For instance, a smartphone user may request to generate detailed geometric data of a nearby building within a matter of milliseconds. Consequently, generating geometric data may take place nearby the end-user, respectively, on an edge device. Thus, evaluating the performance and power consumption aspects of edge devices (i.e., with different hardware architectures) during the execution of procedural geometric workloads is a critical aspect. Therefore, our objective is to comprehend the limitations of different edge devices when performing such workloads with various configurations.

In this paper, we provide a comprehensive performance evaluation of procedural geometric workloads on resource-constrained devices. First, the workloads are packaged into software containers so they can be executed on different devices across the computing continuum. Second, we evaluate the performance and energy consumption aspects of different geometry workloads executed on state-of-the-art resource-constrained devices (i.e., edge devices). Our evaluation findings on a testbed with resource-constrained devices aim to validate the feasibility and applicability aspect to enable the execution of such workloads in proximity to end-users. Furthermore, the results can aid developers and system engineers in designing a system for executing procedural geometry workloads in the computing continuum infrastructures.

The remaining sections are structured as follows. Related work is presented in Section II. Section III discusses latency challenges in 3D visualization and then discusses procedural geometry workloads. Section IV evaluates performance aspects of procedural geometry workloads with different configurations executed on low-powered and resource-constrained devices. Finally, we conclude our discussion in Section V.

II. Related Work

In recent years, modeling and simulating buildings and cities have garnered the academic and industrial sectors' attention. It enables the observation of realistic models in a three-dimensional virtual environment, such as those utilized in simulations to recreate various scenarios. Procedural modeling is one technique to achieve such goals, which requires minimal input to create intricate scenes and structures. When combined with parser generators, it becomes a more efficient and time-saving approach. The resulting application provides users with a quick and straightforward means to generate, model, and design extremely complex structures [14].

Murturi et al. [15] introduce a novel architecture capable of provisioning procedural geometry workloads in edge scenarios. The paper discusses several aspects, including latency and computation requirements, and the role of edge architectures in supporting procedural geometry workloads execution. Greuter et al. [16], propose a framework for the procedural generation of entire virtual worlds in real-time. The paper demonstrates an example of a virtual city populated with building structures, streets, etc. In [17], the authors present an approach to the procedural generation of 'pseudo infinite' virtual cities in real-time. The authors investigated several performance results of the virtual city, building generation algorithm, and LRU (least recently used) cache. Goetz [18] propose a methodology for the automatic generation of highly detailed City Geography Markup Language (CityGML) models by using Volunteered Geographic Information (VGI) from OpenStreetMap (OSM). Kang et al. [19] categorize data for constructing indoor spaces and established standards for the level of detail to provide appropriate application services based on the type and representation method of each data. Kemec et al. [20] propose a conceptual framework for disaster management and a guide to the design, implementation, and integration of the 3D urban modeling tools into disaster risk visualization. In addition, the authors contribute with a new indoor LoD hierarchy for building objects. Furthermore, Hagedorn et al. [21], introduce a classification of indoor objects and structures, and propose a level-of-detail model for the generation of effective indoor route visualization.

Nevertheless, the research works mentioned above focus on solutions for procedural geometry generation techniques, optimizations, automatic generation of highly detailed models, and novel frameworks. However, there has not been any research that investigates performance and energy consumption aspects during the execution of procedural geometry workloads on low-powered and resource-constrained edge devices. Therefore, this paper is a step towards introducing a novel framework that will allow the low-latency generation of 3D content in proximity to end-users and in a distributed manner.

III. Procedural Geometry as a Computational Workload

This section gives detailed information about procedural geometry and explains the main focus of this study. Initially, it provides an overview of 3D visualization and the challenges associated with rendering and latency problems. Subsequently, it briefly delves into the significance of a configurable level of detail (LoD) in procedural geometry workloads.

A. 3D Visualization and Rendering Challenges

Visualizing 3D models with computer graphics has advanced significantly in recent years since numerous techniques have become mature and are being implemented on hardware, meaning that complex 3D scenes can now be seen in real-time on a few hundred dollar game computers, as opposed to requiring a million dollar computer architecture a few years ago [22]. This progress has resulted in a significant demand for more complex and accurate models in 3D content [6]–[8], [23], but the problem is that even though the tools for three-dimensional modeling are becoming increasingly sophisticated, creating realistic models is challenging, time-consuming, and expensive.

According to [24], the 3D graphics industry successfully addresses the majority of issues related to large-scale and dynamic data representations. The methods like real-time rendering and multi-resolution modeling provide the necessary tools and building blocks to develop expressive, effective, and appropriate visual mappings. The research highlights that 3D

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

technology enhances the use of dynamism in visualization. Furthermore, advancements in high-quality real-time rendering techniques have made it possible to create dynamic animated representations and enable interactive exploration to create a virtual reality experience. It is crucial to have the ability to switch 3D viewpoints in real-time in such an environment.

As shown in [25], the simulation and display of a virtual world at interactive frame rates are highly computationally intensive components of a 3D interactive application. Therefore, even with the use of powerful graphics workstations visualizing a complex virtual environment can result in a significant amount of computational load, leading to noticeable lags that can negatively impact display quality. Although rendering images locally results in better responsive user experiences, it usually calls for extensive computing resources that are not always available to the user. This implies that users can leverage the power of cloud computing for computationally intensive rendering tasks; however, client devices (i.e., laptops, tablets, smartphones, etc.) due to enormous data have to deal with latency as a central problem [26].

Fig. 1: CityGML representation example [27].

The term ”latency” denotes the time delay between user input and the corresponding output displayed in the user-facing application. This delay cannot be ignored since the computation of an image is time-consuming in all such systems, and a new image can be generated for each frame, potentially resulting in a system with more than one frame of latency [28]. Usually, this is due to pipelining, which enhances graphics performance which improves graphics performance by processing multiple frames simultaneously (i.e., as the computation for a single frame is split into different stages). Furthermore, due to the high latency brought on by network congestion, 3D representations in navigation apps like Google Earth[1] or 3DCityDB[2] shown in Figure 1, which are designed to assist end-users as they are roaming inside a city are frequently oversimplified, lacking in details and quality [15]. Unfortunately, their 3D representation often lacks in terms of quality due to the vast amount of data that needs to be transferred between devices.

[1]Google Earth, https://earth.google.com/

[2]3DCityDB Database, https://www.3dcitydb.org/3dcitydb/

B. Procedural Geometry Workloads

Procedural Geometry [29] deals with generating geometrical shapes using programming, e.g., a sphere-drawing routine, where the center and radius are given as input. A suitable group of triangles is identified from an algorithm that will be used to define the surface of the sphere. Afterward, the surface serves as an instance of procedural geometry and could be used to draw the sphere in the future. There are various definitions depending on whether geometrical shapes are produced by a pre-processing routine (i.e., such as Terragen [3]), or created in real-time with content lower than the resolution of the raw terrain data.

Objects in three-dimension can be characterized based on the level of detail (LoD) that they use to depict real-world objects. LoD is a key concept to keep in mind when using procedural geometry because it has an impact on the output's complexity, i.e., an object that occupies only 4 pixels in the final image does not need 10,000 polygons; a simple representation using 10 polygons would be sufficient [30]. The use of procedural geometry has proven to be successful in computer graphics and has been advantageous for a variety of reasons, including the ability to quickly and efficiently generate content, therefore saving time.

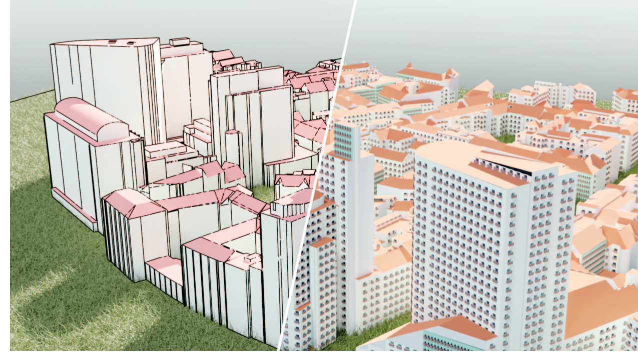

Fig. 2: Procedural generation workload examples with CityGML [15].

Dealing with accurate graphics and visualization applications nowadays requires enormous geometric data consisting of many triangles to be processed; for example, a reasonable LoD of 0.05 km2 near the center of Vienna requires one million triangles, which generates approximately 30 MiB of raw mesh data; whereas it takes less than 0.1 MiB in input parameters to generate raw mesh data for dimensioning, positioning, and orienting each axiom shape, as well as style parameters, which are essential for detailed geometry as is to be seen in Figure 2 where these parameters are extracted and used on CityGML models [15].

Since procedural generation workloads require heavy computation and data transfer, there is a trade-off for transferring the entire 3D data equivalently. This indicates that one may dynamically compute a certain level of detail depending on

[3]Terragen, https://planetside.co.uk/

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

computation or time budgets [15]. The computation can occur in any of the entities across computing continuum infrastructures (i.e., edge, fog, or cloud). Therefore, it is possible to optimize computation and data transferring aspects; however, this requires novel management techniques that overcome such challenges, e.g., the generated mesh data for areas that end-users frequently request can be cached and reused.

Within this paper, we do not address technical details on how models are created or rules applied (i.e., see [31]). This paper only evaluates procedural geometry workloads with different parameter configurations and investigates their execution performance on low-powered and resource-constrained devices (discussed in Section IV).

IV. Evaluation

This section will serve as the foundation for this study by evaluating performance aspects of procedural geometric workloads on resource-constrained devices. To ensure an accurate evaluation, different types of devices will be tested using the same set of cases. This approach will allow an appropriate assessment under which device-specific issues can also be identified. Therefore, the section begins by explaining the execution and configuration details of procedural geometric workloads. It then provides detailed information about testbed architecture before moving on to evaluation methodology and test scenarios. Finally, it finishes with an evaluation of the devices by analyzing data and visualizing graphics about CPU and memory usage, execution time, and power consumption.

A. Execution and Configuration Details

Creating an appropriate encapsulation of procedural generation that can run on heterogeneous hardware platforms is challenging. First, the procedural generation mechanism is implemented in Python and C++. The mentioned mechanism is developed as a client-server model, i.e., a client with a specific configuration request edge-server (i.e., resource-constrained edge devices) for generating 3D visualization. Afterward, the procedural generation mechanism is containerized such that it can be executed in heterogeneous devices across the device-to-cloud continuum. The following command in Listing 1 enables building the mesh generator:

```
run python3 runcmake.py --num-gcc-threads
$NUM_GCC_THREADS --mesh-gen --build-only
```

Listing 1: Building the mesh generator.

The option *--num-gcc-threads* corresponds to the value for the number of threads that the mesh generator can use, whereas the options *--mesh-gen* and *--build-only* build the executable without running it. Several other configurable variables like edge-server port and host settings are configured beforehand in a *"default.json"* file. Furthermore, several other variables can be configured, such as:

- *num_worker*, which specifies the number of threads for the mesh generator edge-server. This variable is labeled in edge-server output as *num_server_threads*,
- *margin*, which for a value of 1, means that the meshes for nine slices (one slice for the query position and eight surrounding slices) will be generated and sent to the client. Each new request may overwrite this value, and the number of slices is calculated as follows: *(margin x 2 + 1) x (margin x 2 + 1)*. If the query position is close to the boundary, the resulting slices will be fewer,
- *LoD* means the default level of detail value, which can also be overwritten in each request and corresponds to the geometric details of the model, e.g., LoD = 2 results in building walls with windows, and
- *session_queue_max_mebibytes* corresponds to the maximum mebibytes (MiB) that can be used by a single HTTP session queue. The default value is 64 (approximately the size of serialized mesh data for a dense area with a margin of 10), which is a soft limit. It is still possible to exceed 64 for the size of the queue if the generated mesh for a single request is more than 64.

Once the container is running in the targeted resource-constrained device, the initial output of the procedural generation mechanism gives information about the coordinates of *city lower* and *city upper* (see Listing 2). The following information is important since it shows the range of coordinates that a client can request for 3D visualization. Therefore, coordinates in a request from the client have to be inside this range; otherwise, the mesh data will not be generated, and the client will get a response with an error message. It's also likely that the query point is out in the middle of a barren area with no surrounding buildings. However, note that the generated mesh will be empty in this scenario.

```
run CPU mesh generator (Release)
log_dir: "/opt/ct2020/data/log"
ct_data_dir: "/opt/ct2020/data"
city  lower: (-10749.2695312, 331344.71875)
city  upper: (18183.7949219, 353381.9375)
city center: (3717.26269531, 342363.3125)
city   size: (28933.0644531 x 22037.21875)
regular  tile dim: 2000 x 2000
regular slice dim: 25 x 25
tiles  grid dim: 15 x 12
slices grid dim: 80 x 80
hash code: 14872646704247140986
http session queue max mebibytes: 64
default margin: 1
default lod: 2
num_pgg_cpu_threads: 4
num_server_threads: 24
listening at: 192.168.110.1:34567
```

Listing 2: An example of mesh generator execution.

ct_data_dir and *log_dir* give the paths of where the mesh generator data is taken, respectively, where the logs will be written if *--log-to-file* is set to true. The output then gives information about *regular tile dim*, which denotes the dimension of each regular tile in meters. On the other hand,

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

regular slice dim stands for the dimension of each regular slice in meters within a regular tile. The dimension of the regular tile grid is represented by *tiles grid dim* and calculated by the total number of regular tiles: 15 x 12, which corresponds to 180 *.rtile* files in the directory *ct2020-vol/rtiles*, whereas *slices grid dim* indicates the dimension of the regular slice grid in each tile. The integrity of data is verified with *hash code*, which is computed from *city lower*, *city upper*, *regular tile dim*, and *regular slice dim* and written into each *.rtile* file and in *rtiles.conf*.

The client can test the performance by sending an HTTP GET request method to an edge-server, e.g., with a curl over the terminal with *margin=2* and *lod=1*, as presented in Listing 3:

```
curl -X GET "192.168.110.1:34567/ctgml/
${x-coord}/${y-coord}?margin=2&lod=1"
--output ctmesh.bin
```

Listing 3: Client request command.

As aforementioned, *x-coordinate* and *y-coordinate* have to be in the range of *city lower* and *city upper*. The option *--output* is used to save the output to a file named *ctmesh.bin*. An edge-server responds to these requests by printing different execution time data on the terminal or writing them to a file, depending on the logging configuration. A Python script is used for sending multiple requests to an edge-server and creating sophisticated tests to evaluate the performance of the devices. To evaluate the performance aspects, the mesh generator is deployed and runs on each resource-constrained edge device (see Table I).

B. Testbed and Methodology

This study conducts a comprehensive performance evaluation in a test environment of four representative resource-constrained devices as shown in Table I. As can be noted, devices are with different architectures ranging from Intel to ARM processors operating in the 64-bit mode. With CPU speeds ranging from 1.2 GHz to 3.4 GHz, these devices will provide a representative evaluation of performance and a variety of results. Each of them runs on Linux-based operating systems with Docker[4] and Python installed. Furthermore, devices are placed in proximity to end-users (i.e., clients that request 3D visualization), meaning that clients and devices are under the same administrative domain and connected via wireless.

Device	Processor	CPU	RAM
RPi3b	ARM Cortex-A53	up to 1.2 GHz	1 GB
RPi4b	ARM Cortex-A72	up to 1.5 GHz	4 GB
Xavier™ NX	Carmel ARM® v8.2	up to 1.9 GHz	8 GB
Lenovo T540p	Intel® Core™ i7-4700MQ	2.4 - 3.4 GHz	8 GB

TABLE I: Testbed overview of representative resource-constrained edge devices.

In order to test whether the earlier mentioned trade-off between low-latency and high-computational devices is worth it, it is of significant interest for this study to see how well these devices support the execution of the workload in terms of execution time, memory usage, CPU usage, and power consumption.

Each device has unique specifications, as described in the subsection above. Additionally, they differ in how they are powered, resulting in different ways of measuring power. Raspberry Pi 3 (RPi3) and Raspberry Pi 4 (RPi4) are powered by a micro-USB power supply, while NVIDIA® Jetson Xavier™ NX and Lenovo T540p use AC adapters. However, the differences do not mean that each device should have different test scenarios. Instead, the goal is to assess how these differences with the same test scenarios affect the results. As concluded in [32], performance-efficient CPUs, like Intel processors, have shorter execution times compared to low-power devices. Still, their power consumption is high, whereas ARM processors, on the other hand, consume less power. The former is used more in embedded systems and IoT devices, while the latter is in personal computers.

The test cases are designed to push devices to their limits and overload them, e.g., evaluating performance aspects of an RPi3 with various configurations of procedural geometry generations. To achieve this, variables such as LoD, which specifies the level of detail, and *margin*, which generates the mesh slices, can be configured. These two parameters are included in an HTTP GET request sent to an edge-server, which impacts execution time and other metrics.

C. Test Scenarios

For each device in the testbed, we execute 100 test cases with a *margin* value ranging between 1-10 and for each LoD value between 1-3. The margin begins with value 1 and increases progressively to 10 (i.e., a set of 10 tests is executed per margin). Increasing these configurable parameters results in a higher load on resource-constrained devices that execute the mesh generator mechanism. The x and y coordinates are generated in different combinations, all in the range of the *city lower* and *city upper* parameters. In the following, we present possible workload configurations in terms of LoDs and margins:

- **margin 1, LoD 1** - a light test in terms of overloading the system with the level of detail 1, meaning that walls are simply rectangles, and margin 1 indicates generating meshes for nine slices.
- **margin 7, LoD 2** - a moderate test with a higher level of detail and margin than the first test. The value of LoD 2 means that walls have windows, and the margin 7 builds the meshes with *(7 x 2 + 1) x (7 x 2 + 1)* = 225 slices.
- **margin 10, LoD 3** - test case that overloads the system and qualifies as a stress test for this framework. Level of detail 3 implies that a more detailed visualization is represented than in the two first test cases, whereas margin 10 indicates that 441 slices are necessary for building meshes.

Figure 3 shows an illustration of a geometry generated and delivered to the end user starting from LoD 0 (left) to the

[4]Docker, www.docker.com

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

Fig. 3: An example of procedural geometry workload. Starting from a CityGML baseline model with LoD: 0 (left) to the detailed geometry with LoD: 3 (right).

detailed geometry with LoD 3 (right). Notice that LoD 0 serves as a CityGML baseline model (i.e., typically a flat polygon) upon which detailed geometries can be generated.

D. Results

In this subsection, we present in detail evaluation results regarding 1) CPU usage, 2) Memory usage, 3) Execution time, and 4) Power consumption. We measure performance stats (i.e., CPU and Memory usage) by executing the "docker stats" command [5] . The CPU usage values are displayed in percentage, which is a common method for measuring this parameter, whereas the memory usage values are displayed in Mebibytes (MiB). The power consumption is measured by using three hardware and software tools. These tools include UM25C USB Meter from Rui Deng [33] for both Raspberry Pis, Tegrastats [6] for NVIDIA, and Powerstat [34] for Lenovo. The values are displayed in Watts (W) and indicate the power consumption of the entire device during execution time. To obtain consistent results, for each result presented in the following graphs, we calculated the 90 percent confidence interval of means.

1) CPU usage: In Figure 4, we present results for CPU usage during the execution of geometry workloads. As can be noted, each of the visualized subgraphs has a specific LoD value and margins ranging from 1 to 10. With a higher margin, the device is expected to experience an increased CPU load since it generates many slices to build the mesh. In Figure 4a, we present results for CPU usage during the execution of geometry workloads with LoD 1. Due to its lower processing and memory capabilities compared to other devices, the RPi3b device is considered to be the "weakest device" in terms of technical specifications. The overall CPU utilization is around 20%, with the highest value being 95% which occurred in the middle of the execution with LoD 3. The RPi4 performed slightly better CPU utilization than RPi3, while surprisingly, the NVIDIA Jetson device showed marginally higher CPU utilization than other devices. Lenovo T540p provides the best performance in terms of CPU usage, with values mainly falling under 5%. In Figure 4b, we present results for CPU usage during the execution of geometry workloads with LoD 2. Lenovo T540p provides the best performance in terms of CPU usage, with values mainly falling under 10%, whereas other devices deliver almost consistent values, with Raspberry Pi 4 performing slightly better. In Figure 4c, we present results for CPU usage during the execution of geometry workloads with LoD 3. Similarly, Lenovo T540p provides the best performance in terms of CPU usage, whereas other devices deliver almost consistent values. These results indicate that devices near end users can execute such workloads, especially with LoD 1 & 2. However, with LoD 3, some devices could experience high overload and not be feasible for executing geometry workloads.

2) Memory usage: In Figure 5, we present results for memory usage during the execution of geometry workloads with different configurations. Regarding memory usage, the results are displayed in Mebibytes (MiB). As shown in Figure 5a, the memory usage of devices with LoD 1 lies mainly under 200 MiB. As LoD increases in Figure 5b, memory usage increases as well, potentially exceeding 1000 MiB, as shown with NVIDIA Jetson. Figure 5c shows that devices use almost consistent memory values, while RPi3 uses slightly less memory since the maximum memory size is 1GB; however, this affects the overall time required to execute geometry workloads. Similar to CPU usage, when increasing LoD, some devices may encounter significant overloading while performing these workloads. Nevertheless, the results indicate that devices near end users can execute such workloads with a specific range of coordinates that a client request for 3D visualization.

3) Execution time: In Figure 6, we assess the time complexity to execute geometry workloads with different configurations. In Figure 6a, we present results of the time complexity to execute geometry workloads with LoD 1. As can be noted, the RPi3b generates 3D content around 0.5 seconds and a maximum of 1.9 seconds. Lenovo T540p shows significantly better processing time compared to other devices. In Figure 6b, we present results of the time complexity to execute geometry workloads with LoD 2. As can be noted, the RPi3b took around 3.2 seconds and a maximum of 13.5 seconds to generate 3D content. In Figure 6c, we present results of the time complexity to execute geometry workloads with LoD 3. The NVIDIA Jetson device performed slightly better in the execution of workloads compared to previous results. The RPi4 device generates 3D content in around 1.5 seconds and a maximum of 3.2 seconds. In all test cases, Lenovo T540p performed better since it has more processor cores compared to other devices. Consequently, this affects the overall power consumption required to execute geometry workloads. Nevertheless, the results indicate that RPi3 and NVIDIA Jetson compared to other devices, may not be suitable for low-latency 3D content generation. However, since there is a trade-off between energy consumption and time requirements, we argue

[5]Docker Stats, https://docs.docker.com/
[6]Tegrastats utility - NVIDIA, https://docs.nvidia.com

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

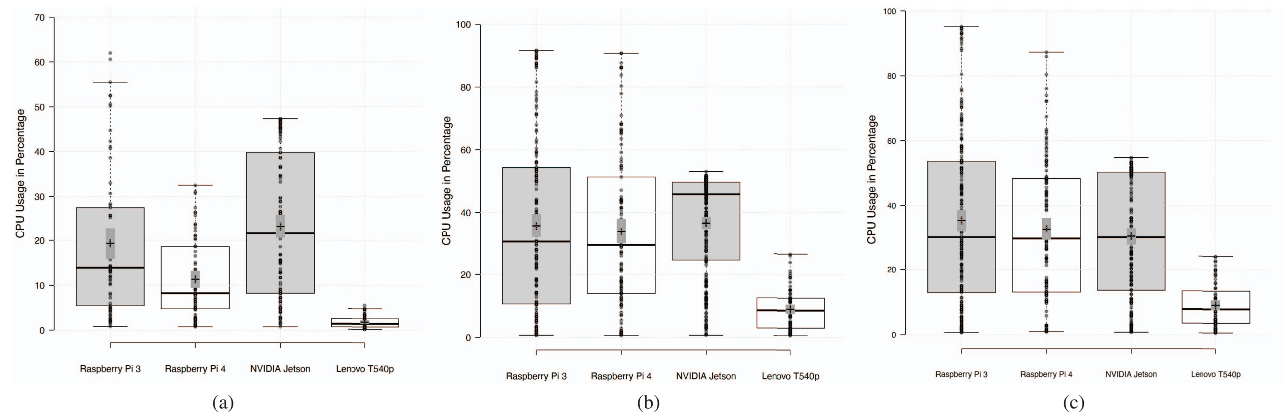

Fig. 4: Analysis of the CPU utilization on edge devices during the execution of geometry workloads with: (a) LoD 1, (b) LoD 2, and (c) LoD 3.

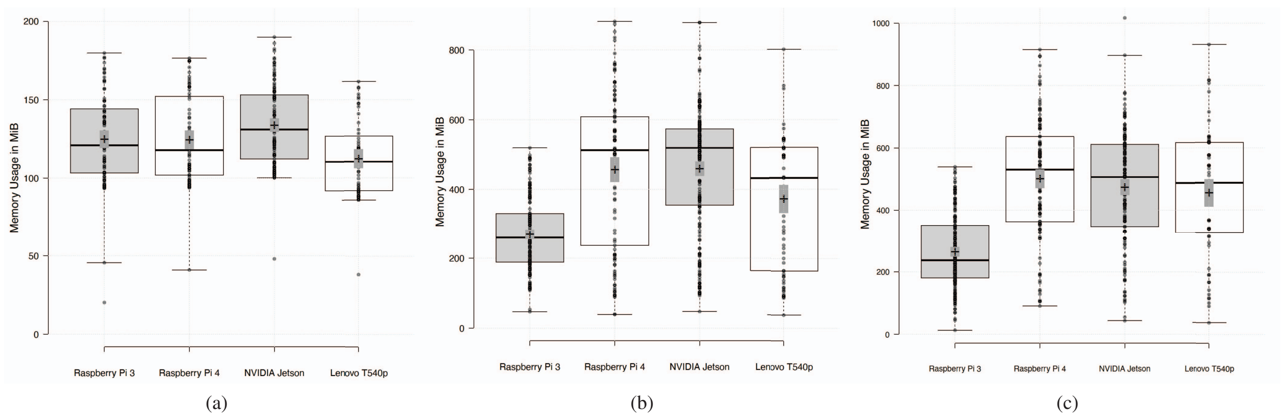

Fig. 5: Analysis of the memory utilization on edge devices during the execution of geometry workloads with: (a) LoD 1, (b) LoD 2, and (c) LoD 3.

that both devices are feasible to execute geometry workloads; however, this entirely depends on 3D generation requirements.

4) Power consumption: In Figure 7, we show an analysis of the power consumption on edge devices during the execution of geometry workloads in different configurations. In Figure 7a, the RPi3b uses minimum power consumption while executing workloads with Lod 1. The power consumption is within the range of 2.5W and almost 3W, where the maximum stands at 4.2W and the minimum at 1.6W. Similar power consumption uses RPi4. At the same time, NVIDIA has shown slightly higher power consumption. As expected, Lenovo T540p has the highest power consumption due to the number of cores used while executing the workloads. In Figure 7b, the power consumption on edge devices running workload with LoD 2 remains almost similar to the previous graph, except for the Lenovo T540p, which doubles power consumption by showing values in the average of 9.6W and reaching a maximum of 30W. This aligns with earlier results regarding CPU, memory, and time required to execute workloads. In Figure 7c, the power consumption on edge devices running workload with LoD 3 remains almost similar to the previous graph in Figure 7b. Nevertheless, the power consumption results indicate the applicability and feasibility to execute geometry workload on low-powered and resource-constrained edge devices. It is also worth noting that the visualization of 3D content on edge devices occurred within the range of the *city lower* and *city upper* parameters mentioned in Section IV-C.

Lastly, Lenovo T540p demonstrated satisfactory performance. Additionally, the RPi4 delivered noteworthy results. Based on this study, it is also observable that when increasing the level of detail and increasing the margin to generate more mesh slices, the RPi3 may become unreliable since the values provided, e.g., execution time, might sometimes be excessive.

E. Limitations

In this study, procedural geometric workloads are solely executed on CPU hardware. This means that the procedural geometry workloads were executed exclusively by the central

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

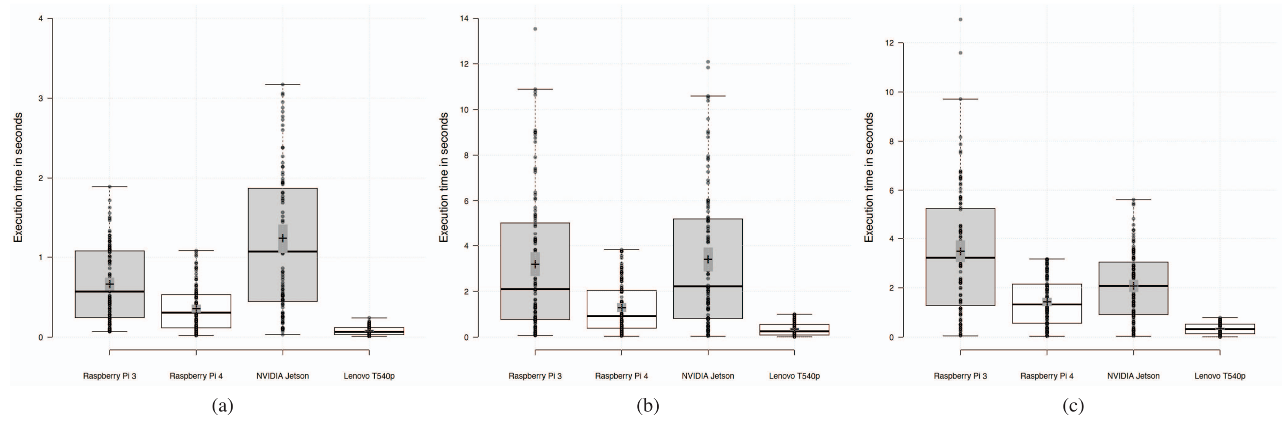

Fig. 6: Time required to execute geometry workloads with: (a) LoD 1, (b) LoD 2, and (c) LoD 3.

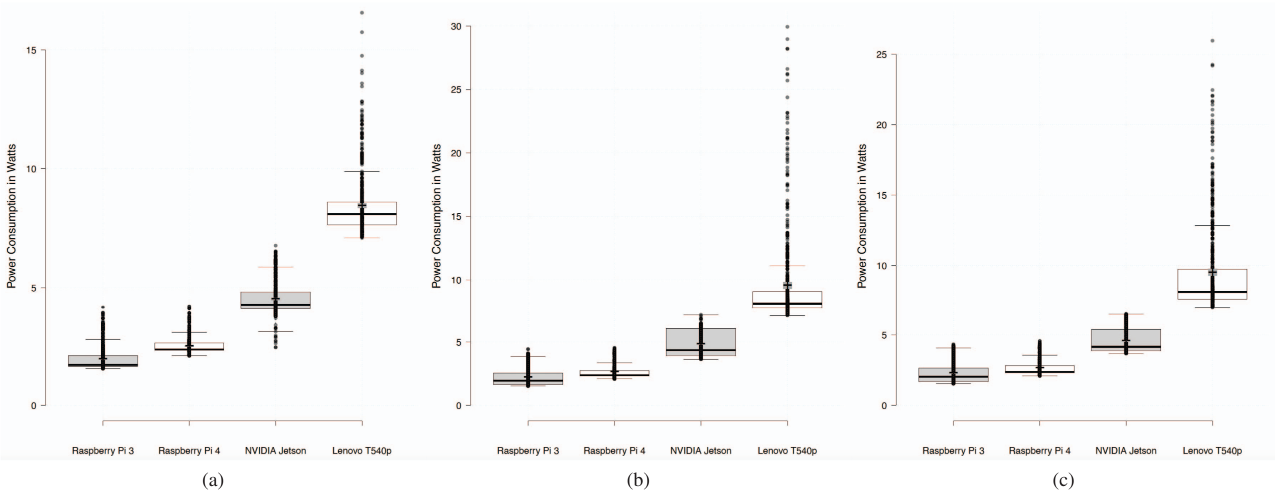

Fig. 7: Analysis of the power consumption on edge devices during the execution of geometry workloads with: (a) LoD 1, (b) LoD 2, and (c) LoD 3.

processing unit without using the graphic processing unit (GPU). GPU-based execution [35] would significantly improve overall execution time results (i.e., especially in the case of the NVIDIA Jetson device since it has a GPU with 48 tensor cores). Therefore, we acknowledge such a limitation and must be carefully addressed in future work.

V. CONCLUSION

This research paper provided a comprehensive performance evaluation to execute procedural geometric workloads in low-powered and resource-constrained edge devices. Through extensive experiments, we showed and argued the applicability and reliability of executing such workloads in proximity to end-users, respectively, in edge devices. Despite the promising results, this paper is only a small step towards implementing the operationalization of a framework aiming to achieve efficient provisioning procedural geometry workload on computing continuum infrastructures. Therefore, we aim to provide a complete architectural framework and address technical aspects in our future work.

ACKNOWLEDGMENT

Research has partially received funding from SmartCT and from grant agreement No. 101079214 (AIoTwin). We thank the Institute of Visual Computing & Human-Centered Technology, Research Unit of Computer Graphics (TUW, Austria) for the Procedural Geometry Workload development during the SmartCT project.

REFERENCES

[1] S. Dustdar and I. Murturi, “Towards distributed edge-based systems,” in *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 1–9, IEEE, 2020.
[2] V. Casamayor Pujol, A. Morichetta, I. Murturi, P. Kumar Donta, and S. Dustdar, “Fundamental research challenges for distributed computing continuum systems,” *Information*, vol. 14, no. 3, p. 198, 2023.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.

[3] V. Casamayor Pujol, P. K. Donta, A. Morichetta, I. Murturi, and S. Dustdar, "Distributed computing continuum systems–opportunities and research challenges," in *Service-Oriented Computing–ICSOC 2022 Workshops: ASOCA, AI-PA, FMCIoT, WESOACS 2022, Sevilla, Spain, November 29–December 2, 2022 Proceedings*, pp. 405–407, Springer, 2023.

[4] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[5] I. Murturi and S. Dustdar, "A decentralized approach for resource discovery using metadata replication in edge networks," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2526–2537, 2021.

[6] C. A. Vanegas, D. G. Aliaga, B. Benes, and P. Waddell, "Interactive design of urban spaces using geometrical and behavioral modeling," *ACM Trans. Graph.*, vol. 28, no. 5, p. 111, 2009.

[7] D. Hildebrandt and R. Timm, "An assisting, constrained 3d navigation technique for multiscale virtual 3d city models," *GeoInformatica*, vol. 18, no. 3, pp. 537–567, 2014.

[8] V. Heuveline, S. Ritterbusch, and S. Ronnas, "Augmented reality for urban simulation visualization," *Preprint Series of the Engineering Mathematics and Computing Lab*, no. 16, 2011.

[9] E. Visconti, C. Tsigkanos, Z. Hu, and C. Ghezzi, "Model-driven engineering city spaces via bidirectional model transformations," *Software and Systems Modeling*, pp. 1–20, 2021.

[10] J. Kim, H. Kavak, and A. Crooks, "Procedural city generation beyond game development," *ACM SIGSPATIAL Special*, vol. 10, no. 2, pp. 34–41, 2018.

[11] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 614–623, 2006.

[12] F. Biljecki, J. Zhao, J. Stoter, and H. Ledoux, "Revisiting the concept level of detail in 3d city modelling," in *8th 3DGeoInfo Conference & WG II/2 Workshop, Istanbul, Turkey, 27–29 November 2013, ISPRS Archives Volume II-2/W1*, ISPRS, 2013.

[13] M. Steinberger, M. Kenzel, B. Kainz, J. Müller, P. Wonka, and D. Schmalstieg, "Parallel generation of architecture on the GPU," *Comput. Graph. Forum*, vol. 33, no. 2, pp. 73–82, 2014.

[14] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 669–677, 2003.

[15] I. Murturi, C. Jia, B. Kerbl, M. Wimmer, S. Dustdar, and C. Tsigkanos, "On provisioning procedural geometry workloads on edge architectures.," in *WEBIST*, pp. 354–359, 2021.

[16] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Undiscovered worlds–towards a framework for real-time procedural world generation," in *Fifth International Digital Arts and Culture Conference, Melbourne, Australia*, vol. 5, p. 5, 2003.

[17] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Real-time procedural generation ofpseudo infinite'cities," in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pp. 87–ff, 2003.

[18] M. Goetz, "Towards generating highly detailed 3d citygml models from openstreetmap," *International Journal of Geographical Information Science*, vol. 27, no. 5, pp. 845–865, 2013.

[19] H.-Y. Kang and J. Lee, "A study on the lod (level of detail) model for applications based on indoor space data," *Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography*, vol. 32, no. 2, pp. 143–151, 2014.

[20] S. Kemec, S. Zlatanova, and S. Duzgun, "A new lod definition hierarchy for 3d city models used for natural disaster risk communication tool," in *Proceedings of the 4th International Conference on Cartography & GIS, Volume 2, Albena, June 2012, pp. 17-28*, International Cartographic Association, 2012.

[21] B. Hagedorn, M. Trapp, T. Glander, and J. Döllner, "Towards an indoor level-of-detail model for route visualization," in *2009 tenth international conference on mobile data management: systems, services and middleware*, pp. 692–697, IEEE, 2009.

[22] M. Pollefeys, "Visual 3d modeling from images.," in *VMV*, pp. 11–12, 2004.

[23] T. Jund, P. Kraemer, and D. Cazier, "A unified structure for crowd simulation," *Comput. Animat. Virtual Worlds*, vol. 23, no. 3-4, pp. 311–320, 2012.

[24] J. Wood, S. Kirschenbauer, J. Döllner, A. Lopes, and L. Bodum, "Using 3d in visualization," in *Exploring geovisualization*, pp. 293–312, Elsevier, 2005.

[25] T. K. Heok and D. Daman, "A review on level of detail," in *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*, pp. 70–75, IEEE, 2004.

[26] R. Cloete and N. Holliman, "Measuring and simulating latency in interactive remote rendering systems," vol. abs/1905.05411, 05 2019.

[27] "The citygml database, https://www.3dcitydb.org/3dcitydb."

[28] M. Olano, J. Cohen, M. Mine, and G. Bishop, "Combatting rendering latency," in *Proceedings of the 1995 symposium on Interactive 3D graphics*, pp. 19–ff, 1995.

[29] M. Widmark, "Procedural geometry for real-time terrain visualisation in blueberry3d," in *Digital Earth Moving*, pp. 43–47, Springer, 2001.

[30] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *ITB Journal*, vol. 14, no. 3, pp. 342–351, 2006.

[31] C. Jia, M. Roth, B. Kerbl, and M. Wimmer, "View-dependent impostors for architectural shape grammars," in *Pacific Graphics Short Papers, Posters, and Work-in-Progress Papers*, pp. 63–64, Eurographics Association, 2021.

[32] M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry, "Performance evaluation and energy efficiency of high-density hpc platforms based on intel, amd and arm processors," in *European Conference on Energy Efficiency in Large Scale Distributed Systems*, pp. 182–200, Springer, 2013.

[33] "Ruideng um25c manual, [online]. https://www.manualslib.com/products/ruideng-um25c-10243666.html."

[34] Canonical, "Powerstat - a tool to measure power consumption, [online]. https://manpages.ubuntu.com/manpages/xenial/man8/powerstat.8.html."

[35] M. Steinberger, M. Kenzel, B. Kainz, P. Wonka, and D. Schmalstieg, "On-the-fly generation and rendering of infinite cities on the gpu," in *Computer graphics forum*, vol. 33, pp. 105–114, Wiley Online Library, 2014.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 10,2023 at 14:44:48 UTC from IEEE Xplore. Restrictions apply.