Operating systems benchmarking at generation of physically based rendered images

Andrey Arguedas Espinoza
Computer Science Engineering School
Costa Rica Institute of Technology
San Jose, Costa Rica
and12rx12@gmail.com

Michael Chen Wang
Computer Science Engineering School
Costa Rica Institute of Technology
San Jose, Costa Rica
michael.cw02@gmail.com

***Abstract*—In this paper, we compare operating systems based on their performance at generating physically based rendered images, we do it by designing an experiment focused on benchmarking the performance of the operating systems. Our main objective is to determine whether there is a performance improvement between operating systems running natively compared to virtual machines or subsystems. With our experiment we found that there are significant statistical difference between different operating systems and configurations for rendering physically based images, and also there is a few which the difference is very small. The resources of our experiments can be found here: https://gitlab.com/itcr-doe-2023/proyecto-2-pbrt.**

***Index Terms*—Benchmarking, operating systems, Physically Based Rendered Images, Virtual Machines, Performance, ANOVA, Post-Hoc, Pairwise.**

I. Introduction

Opinions regarding the performance of operating systems vary widely, in our careers on Computer Science we have heard opinions of people who believe that certain operating systems are superior to others depending on their tasks or overall experience with them. In this paper, we aim to investigate whether there is a statistically significant difference between operating systems in the process of render physically based images. To accomplish this, we will utilize the Physically Based Rendering Toolkit - **PBRT** Version 3. PBRT is a software application capable of generating photo-realistic images based on principles of physics and mathematics that simulate real-world light effects [1].

Our main idea is to generate a set of physically based images with different configurations using five different operating systems to render them and compare the time that each operating system takes to successfully generate the images. The configurations that are going to be used for this experiment is by changing the rendering options parameters **Accelerator** and **Sampler** from PBRT. The operating systems in which the scenes are going to be tested on are: Windows 11 installed natively, Ubuntu 22.04 installed natively, Ubuntu as Virtual Machine (WSL) with Windows as host, Windows Subsystem Linux and FreeBSD.

We think that rendering images is a good task for doing these kind of benchmarks since we are consuming a lot of CPU at generate these type of images [1] (we are not using GPU for this experiment), also it will be very important to create an experiment that will work with this kind of problems, it means that we need to review that the images are being rendered successfully to avoid common issues with this type of studies [2] [3].

Our main purpose is to have actual scientific proof of which operating system performs faster in regards of image generation rather than going by "feel".

Our paper is comprised of the following sections: I. Related Work, II. Experiment design, III. Methodology and IV. Results IV. Conclusion

II. Related Work

Benchmarking is a method to analyze and study computer systems performance by studying the execution of benchmark programs to do a comparison between the systems [4]. Regarding related work of operating systems benchmarking we have found different tests, it is possible to study multiple factors like memory consumption, processing performance [5] and running time. Usually the idea is to execute the same tasks in different operating systems and compare the behavior of the study factors [4] [6].

Image Based Rendering is a method to transform a 3D image into a 2D image using geometry figures to approximate the objects' shape, light, and shadow in a 3D scene [1]. Ray Tracing is a technique to generate images shooting rays from a camera and calculating the light based on the objects they meet [7], is a very important process in today's industry, it has applications in movies and animations [8], gaming and interactive graphics [9], and scientific visualization [10].

In all of these areas the time is very important, so the industry has been always trying to reduce rendering time in order to save resources [11]. In the industry it has been always important to do benchmarking of operating systems [12] in order to find the best approach for specific tasks [6]. For this experiment we will be doing benchmarking based on rendering time with the PBRT project which is based on the Physically Based techniques that attempt to simulate reality using principles of physics to model the interaction of light and matter [1] offering superior photo-realism due to its

intrinsic characteristics of reflection, refraction, and shadowing complexity [1].

III. Experiment Design

We will use five combinations of different operating systems implementations. Natively, we have installed Windows 11 Pro, Ubuntu 22.04 and FreeBSD. As Virtual Machines, we have installed Ubuntu 22.04 using VMWare with Windows 11 Pro as host and the Windows Subsystem Linux running on Windows 11 Pro.

All these operating systems have run in the same computer, which have an Intel Core i7-9750H CPU that has 6 cores and 12 threads, 32 GB of RAM, an SSD and a HDD where the natively installed operating systems are located. We will do 5 random raffles to define the order in which the different operating systems will be running our tests I. On each operating system we run a batch of 16 PBRT scenes that will take different amounts of time to be rendered, usually between 5 to 28 minutes, also for each each time that we start to generate a new set of scenes, we will shut down the computer for more than 3 minutes in order to make sure that each batch enters with the computer as "clean" as possible, specially with memory management and CPU heat. We will use 4 base images 1, and generate 4 new images for each original image combining the attributes **Accelerator = kdtree** ,**Accelerator = bvh** , **Sampler = halton**, **Sampler = sobol**. The **Accelerator** will let us modify the type of aggregate to use for efficiently finding ray-shape intersections [1] and the **Sampler** to change the type of Monte Carlo integration for pixels [1], when combining all of this we will get 4 new images with the following configurations:

- **image-kdtree-halton**
- **image-kdtree-sobol**
- **image-bvh-halton**
- **image-bvh-sobol**

This will give us a total of 16 images, which will be rendered one by one on each operating system and collect the rendering time for future analysis. The order of these images will be selected randomly by a script that does raffles for each operating system at a time.

The results of the raffles to see the order of running of operating systems can be seen here I and the raffles of images order in our repository with this **link**. Also you can consult the **Powershell** and **Shell** scripts that we used to automate the data extraction process and save it in the following**text file**.

A. Methodology

First we start doing a raffle to define the random order that we will run PBRT images on each operating system, the result we got from the main raffle can be seen in table I.

After this we created a Python script that will do raffles for the images itself, the script can be found in **here** and the result of images raffles in **here**, we believe this is very important in order to guarantee fully randomized tests and do not induce statistical errors. The automated scripts are in charge to read a

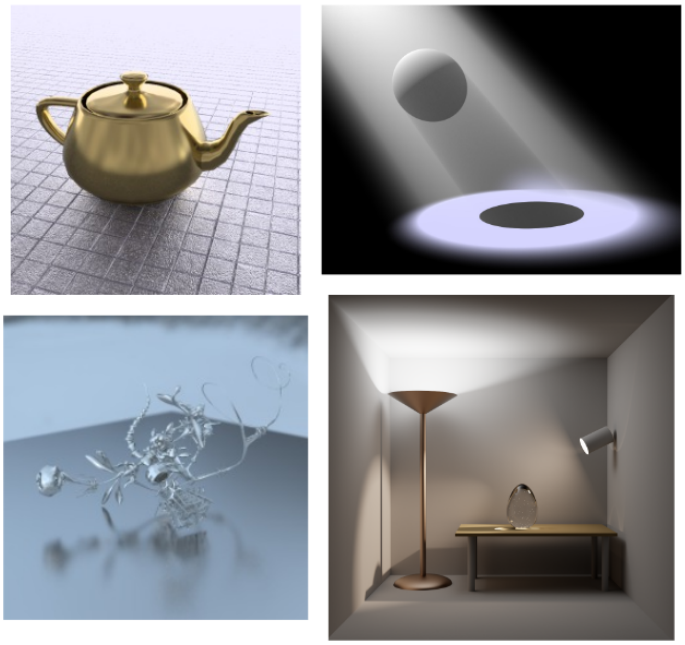

Fig. 1. Original PBRT images

Fig. 2. PBRT image with experiment configurations

	Lot 1	Lot 2	Lot 3	Lot 4	Lot 5
1	Ubuntu	WSL	UbuntuVM	WSL	Windows
2	UbuntuVM	Windows	WSL	Windows	FreeBSD
3	WSL	Ubuntu	Windows	FreeBSD	Ubuntu
4	Windows	UbuntuVM	FreeBSD	Ubuntu	WSL
5	FreeBSD	FreeBSD	Ubuntu	UbuntuVM	UbuntuVM

TABLE I
Order of runs for each Lot. Ubuntu stands for Ubuntu installed natively to save space on the paper.

raffle and start rendering each image, which will save all the metadata with the following order:

- Scene
- Configuration (rendering options parameters)
- Running Time
- operating system
- Lot Number

Since for each original image now we will have 4 configured ones, the original image **bidir** will look like this in the new combinations 2. It is important to understand that the configuration is a 2^k factor, so that is the reason that we generate 16 images in total.

You can check the final results in the following **text file**, also there is a backup of all logs and images generated in **this folder**.

The main factors of this study can be seen in table II, in which our response variable is Time and the most important factor is operating system, then the Configuration and finally the Scene.

Operating System	Configuration	Scene
Windows	kdtree-halton	teapot-metal
Ubuntu	kdtree-sobol	spotfog
UbuntuVM	bvh-halton	buddha-fractal
WSL	bvh-sobol	bidir
FreeBSD		

TABLE II
FACTORS AND LEVELS OF OUR EXPERIMENT

1) Operating system Benchmark: With the data collected we start by analyzing it using the programing language R, first we start by reviewing the data in order to check that everything should be as expected, since we should have 80 images per lot in 5 lots that means 400 images total, also 4 levels of configuration **(bvh-halton, bvh-sobol, kdtree-halton, kdtree-sobol)** see summary of data 3.

```
          Scene          Configuration      Time                OS      Lot
bidir         :100   bvh_halton   :100   Min.   : 137.7   BSD         :80   1:80
buddha-fractal:100   bvh_sobol    :100   1st Qu.: 222.4   UbuntuNative:80   2:80
spotfog       :100   kdtree_halton:100   Median : 311.0   UbuntuVM    :80   3:80
teapot-metal  :100   kdtree_sobol :100   Mean   : 462.5   Windows     :80   4:80
                                         3rd Qu.: 565.1   WSL         :80   5:80
                                         Max.   :1714.4
```

Fig. 3. Summary of Data

After that, we want to view the time box plot of the operating systems, see figure 4, in this chart we can see that all operating systems boxes collide each other, but we see some differences, first we can see the mean (red "+" sign) of Windows 11 is higher compared to the rest and is close to the third quart of the other plot boxes, also we can see that Windows produces some more outliers compared to the other ones. With this plot we can see the range of rendering times through all of our tests, but we can not conclude anything, we need a more in depth analysis.

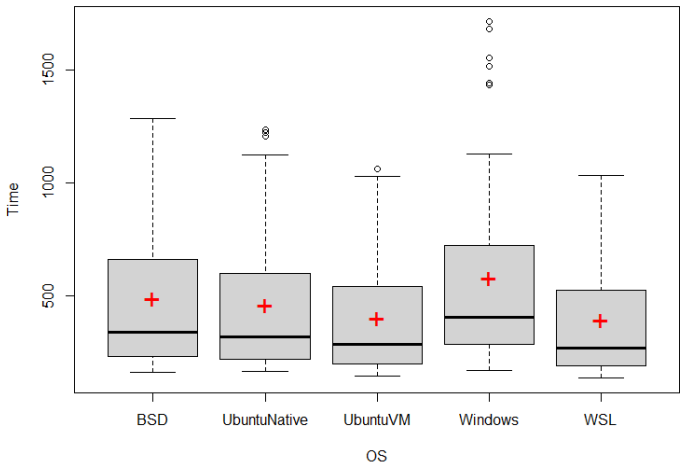

Fig. 4. operating system boxplot by time

Another important factor in our study is the **configuration** of images, is the reason why we are doing a 2^k experiment, in the configuration chart 5 we see that all ranges collides but the configurations with **"Accelerator=kdtree"** has larger range and outliers, so this combinations could be generating the outliers that we see on Windows 11 4. We can see that the configuration of **kdtree** in general looks to be a little bit slower than **bvh**, but in order to reach a conclusion we will need more analysis.

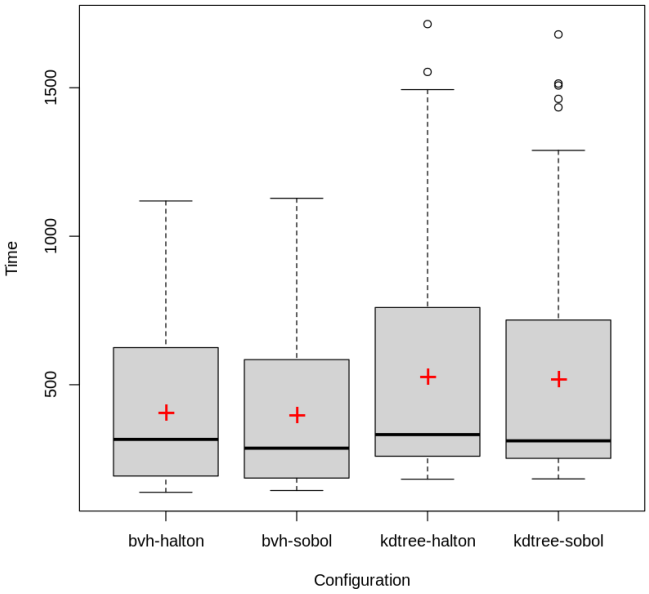

Fig. 5. Modified configuration of PBRT scenes box plot by time

The third factor we need to analyze is the **scene**, in PBRT every scene has a different configuration, shapes and assets, so this is a factor that interacts in our data, see box plot chart 6.

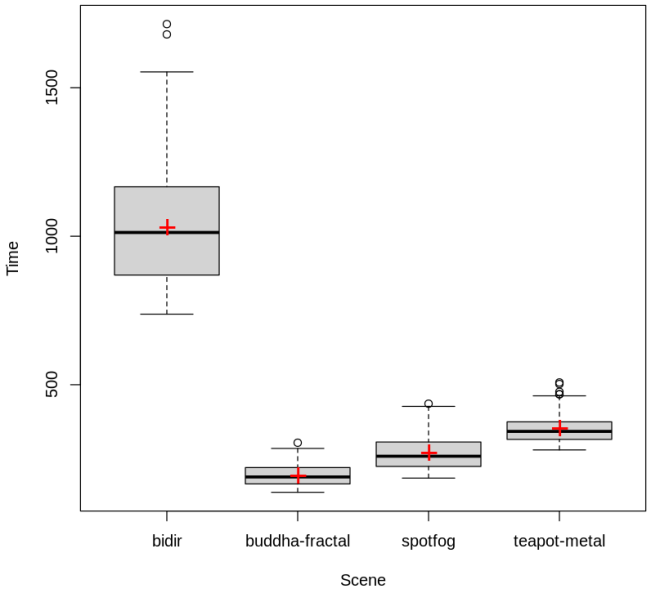

Fig. 6. PBRT scene definition box plot by time

Since we are running 5 different lots, it is important to review the range of the lots by time in order to check that all of them are similar, we would not like to have a lot that is very different to the others because that could mean that there was a problem with our computer or in the way that we run the lot at that specific time (see figure 7).

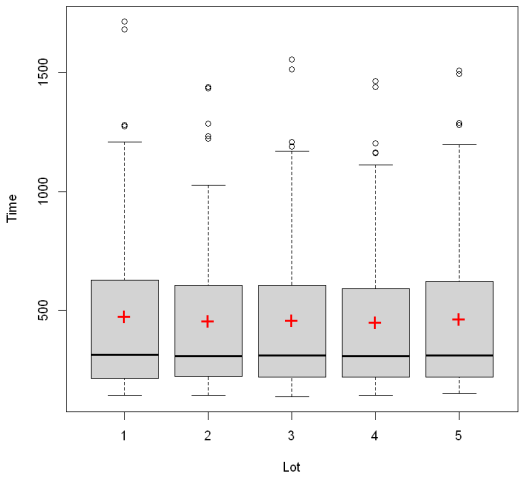

Fig. 7. Box plot of experiments by Lot and Time

B. *Interaction Charts*

For the kind of experiment that we are doing it is needed to see the interaction of the factors in function of the time that the images took to render. First we start by analyzing the **configuration** factor in response of the time 8 , and we can see that across all operating systems we get that the **kdtree** configuration takes in average more time than the **bvh**.

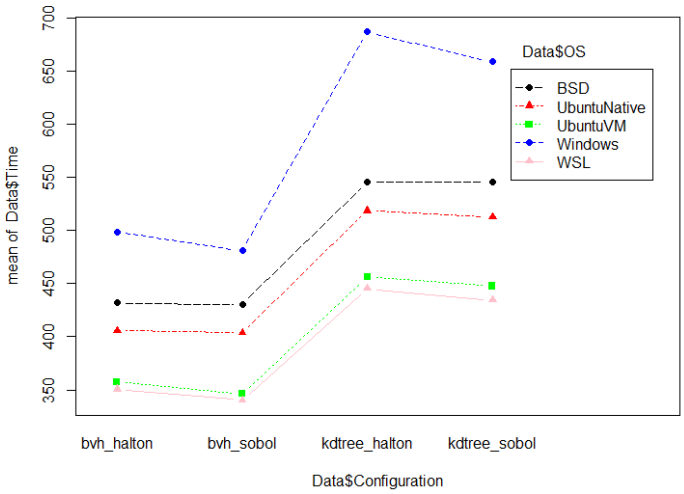

Fig. 8. Interaction of configuration in function of time between operating systems

Since we have different images we need to review the interaction they have with the operating system in the rendering time 9. We can see that for the smaller images all operating systems except for Windows look very similar, and in the biggest image (bidir) windows tends to take more time in average.

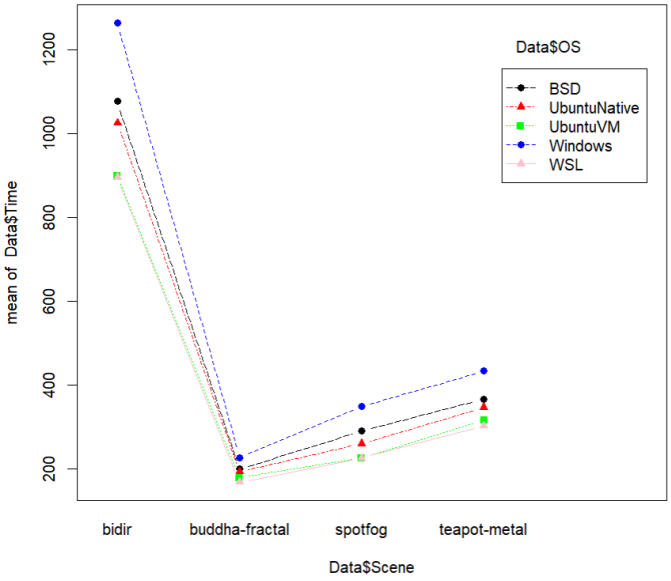

Fig. 9. Interaction of scene in function of time between operating systems

It is important to also see what interaction does the lots have in our response variable 10, since we would not like to see a very different spike in one lot for an operating system. We can see that Windows is the less consistent since it has a few spikes going up and down, while the other have a more linear tendency.

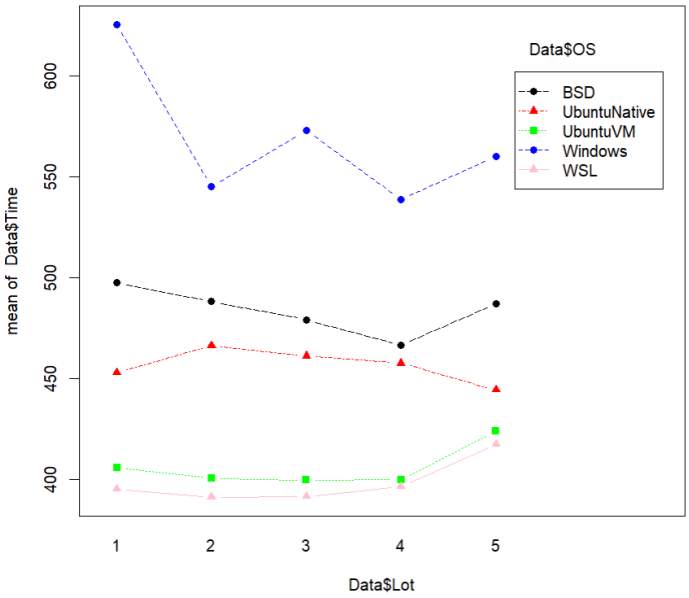

Fig. 10. Interaction of Lot in function of Time between operating systems

C. Data Normality and Homoscedasticity

To check whether the collected data has the bases of homoscedasticity, we chose to do a levene's test with 3 different data transformations on the Time variable (see table III), in which the Square Root scenario yielded a result that is good enough (P-Value: 0.0589) for us to do the lineal model.

Having the results of levene's test, we still wanted to study the ANOVA assumptions of the raw data. So, we created a lineal model with the interactions of $Scene$, $OperatingSystem$ and $Configuration$ for each of the transformations above. In figure 11, is the result of the ANOVA table with the raw data and based on the P-Values, we can see that there are significant statistical differences between the interactions of all factors, all of which have a confidence level of almost 95%.

	Raw Data	Square Root	Cube	Logarithmic
P-Value	4.69e-06	0.0589	0.3475	0.8961

TABLE III
RESULTS OF THE LEVENE TEST AFTER APPLYING SOME BASIC DATA TRANSFORMATIONS

By checking the assumptions of the ANOVA (of the raw data), the residuals histogram looks like it is following normal distribution 12. In the homoscedastic patterns we could see a "cone" pattern from left to right 13 and the normal Q-Q 14 could have a better fit.

```
Anova Table (Type II tests)

Response: Time
                        Sum Sq  Df    F value    Pr(>F)
OS                     1521686   4   484.3812 < 2.2e-16 ***
Configuration          1464837   3   621.7134 < 2.2e-16 ***
Scene                 44509305   3 18890.8601 < 2.2e-16 ***
OS:Configuration         95570  12    10.1406 < 2.2e-16 ***
OS:Scene                775922  12    82.3302 < 2.2e-16 ***
Configuration:Scene    1760583   9   249.0785 < 2.2e-16 ***
OS:Configuration:Scene  136836  36     4.8397 3.234e-15 ***
Residuals               251320 320
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Fig. 11. ANOVA table with original data

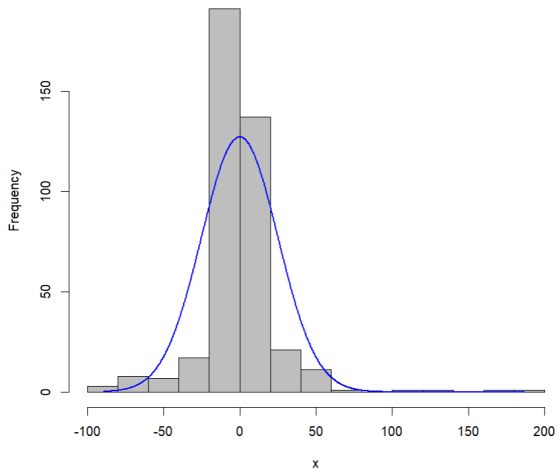

Fig. 12. Residuals histogram

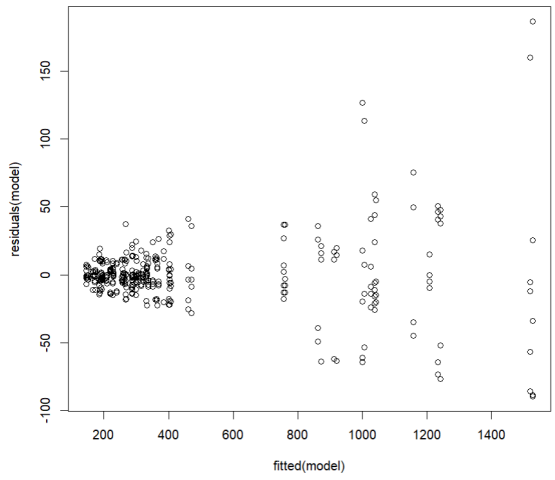

Fig. 13. Homoscedastic pattern of raw data

D. Data Transformation by Square Root

After seeing that the raw data did not fit the ANOVA assumptions perfectly (mainly the Normal QQ chart), we carry on with the square root transformation to all values on the Time variable. In this case, we got a new residuals histogram 15 which looks more "compact" and a better looking

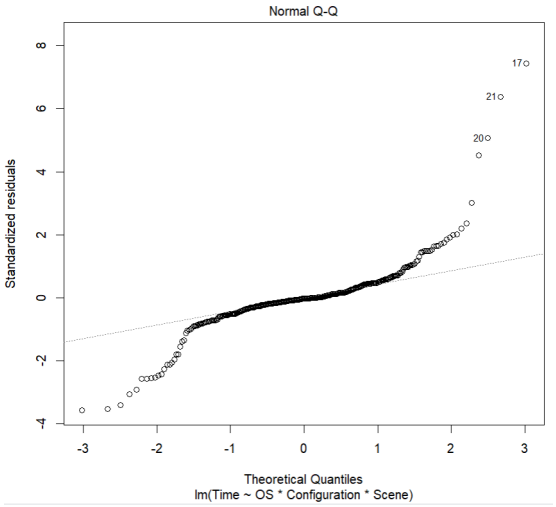

Fig. 14. QQ residuals of raw data

homoscedastic pattern 16, also the normal Q-Q chart 17 looks fitter, and a higher p-value III in the Levene's Test.

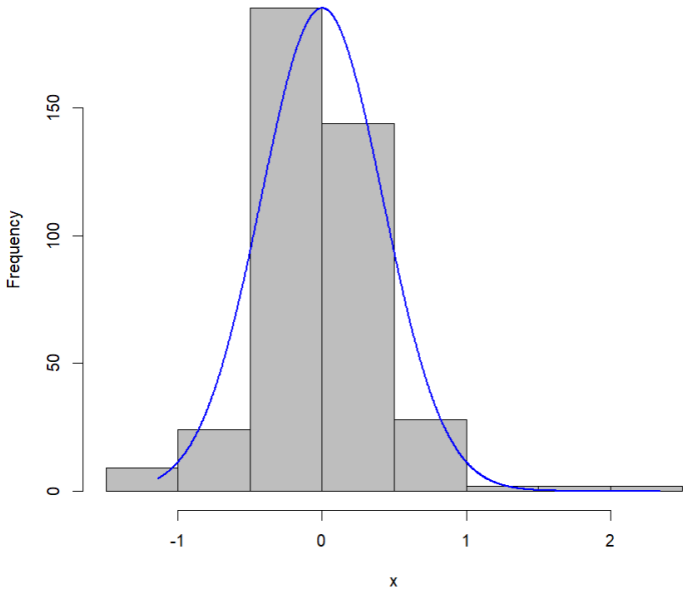

Fig. 15. Residuals histogram after square root transformation

We decided to use the square root transformation because the Levene's test was already good enough with a P-value of 0.05899 and the results from the further transformations didn't look better than the square root one, which is because the ANOVA analysis of variance is permissive of some deviations [13].

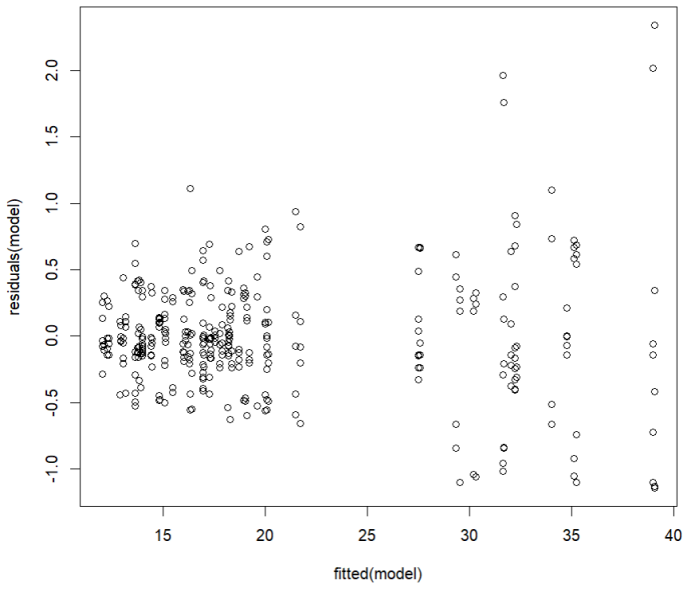

Fig. 16. Residuals after square root transformation

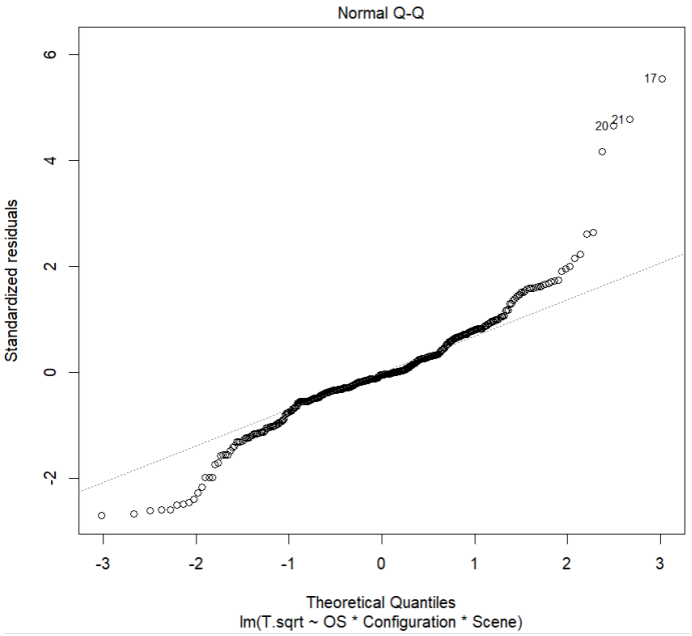

Fig. 17. QQ residuals after square root transformation

With the transformed data we can take a look at the Post-Hoc Analysis, first we did it for the operating system in figure 18 and see that all of the values have significant statistical difference between the groups with a 95% confidence in our experiment.

```
OS           lsmean      SE  df lower.CL upper.CL .group
WSL           18.80 0.05267 320    18.66    18.93  a
UbuntuVM      19.02 0.05267 320    18.88    19.16   b
UbuntuNative  20.12 0.05267 320    19.99    20.26    c
BSD           20.74 0.05267 320    20.61    20.88     d
Windows       22.44 0.05267 320    22.31    22.58      e

Results are averaged over the levels of: Configuration, Scene
Confidence level used: 0.95
Conf-level adjustment: sidak method for 5 estimates
P value adjustment: tukey method for comparing a family of 5 estimates
significance level used: alpha = 0.05
```

Fig. 18. Post-Hoc test between operating systems

In the case of the other factors like *scene* and *configuration* we got very similar results in the Post-Hoc analysis. Even though these are not the main factors of our study, but it still show that it has a significant statistical difference between the scenes and configurations that were chosen (figure 19 and figure 20). With these results, it shows that they need to be taken into consideration along with the operating system factor to study it's interactions.

```
Scene          lsmean      SE  df lower.CL upper.CL .group
buddha-fractal  13.86 0.04711 320    13.74    13.98  a
spotfog         16.35 0.04711 320    16.23    16.46   b
teapot-metal    18.76 0.04711 320    18.64    18.88    c
bidir           31.94 0.04711 320    31.82    32.05     d

Results are averaged over the levels of: OS, Configuration
Confidence level used: 0.95
Conf-level adjustment: sidak method for 4 estimates
P value adjustment: tukey method for comparing a family of 4 estimates
significance level used: alpha = 0.05
```

Fig. 19. Post-Hoc test between scenes

```
Configuration lsmean      SE  df lower.CL upper.CL .group
bvh_sobol      18.86 0.04711 320    18.74    18.97  a
bvh_halton     19.13 0.04711 320    19.01    19.25   b
kdtree_sobol   21.33 0.04711 320    21.21    21.45    c
kdtree_halton  21.58 0.04711 320    21.46    21.70     d

Results are averaged over the levels of: OS, Scene
Confidence level used: 0.95
Conf-level adjustment: sidak method for 4 estimates
P value adjustment: tukey method for comparing a family of 4 estimates
significance level used: alpha = 0.05
```

Fig. 20. Post-Hoc test between scene configuration

IV. Results

With the data transformed we saw that effectively there is a significant statistical difference between operating systems, configurations and scenes because all of them belong to different statistical results. Therefore, we can visualize this with the average rendering time of the transformed data between operating systems 21, configuration 22 and scene 23. From the figure 21, we can gather that overall, Windows seems to be slower than the other operating systems, while WSL and UbuntuVM are the fastest. While figure 22 and 23 are not the objective of this paper, we can see that the accelerator has some effects on the rendering times, with *bvh* being faster for the accelerators and *sobol* being a bit faster than *halton* for the sampler scenario. But this is still not significant enough to make any conclusions, we can also see that there are significant differences between the scenes, which is what we expected.

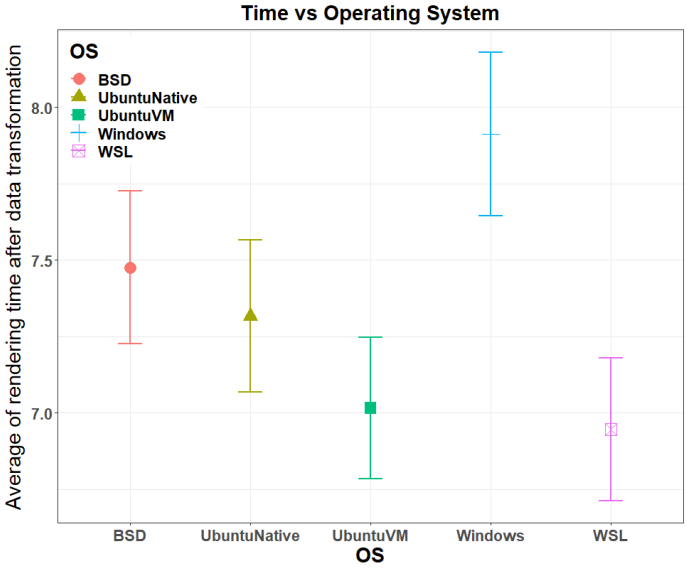

Fig. 21. Average of rendering time after data transformation for Operative System

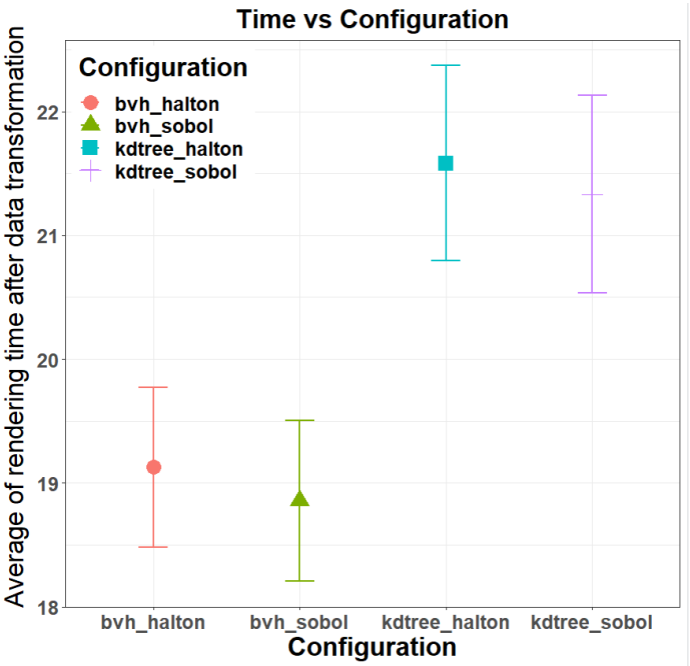

Fig. 22. Average of rendering time after data transformation for configuration

We are also interested in the interactions between factors, where the main ones being the Operating system, Configuration and Scene in figure 24, and we can see that Windows in general has higher (slower) averages than the others. The same can be seen from figure 25, where Windows seems to be significantly slower than the others as well.

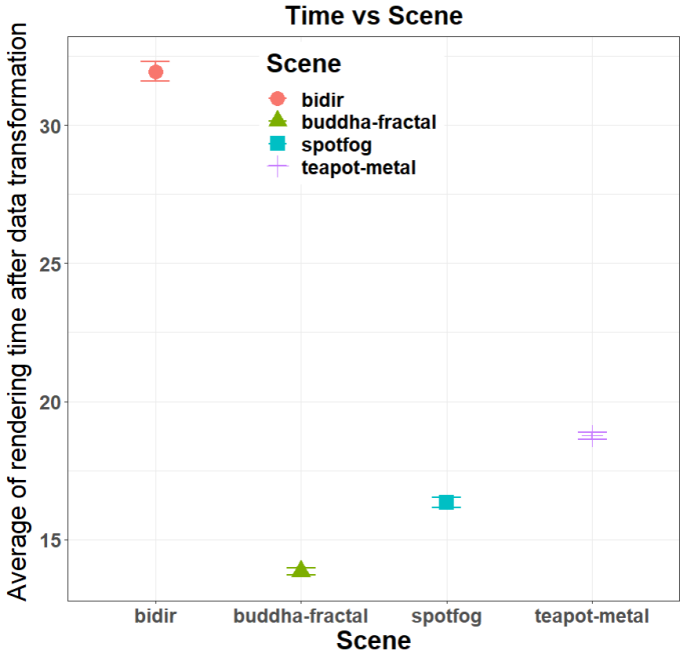

Fig. 23. Average of rendering time after data transformation for scene

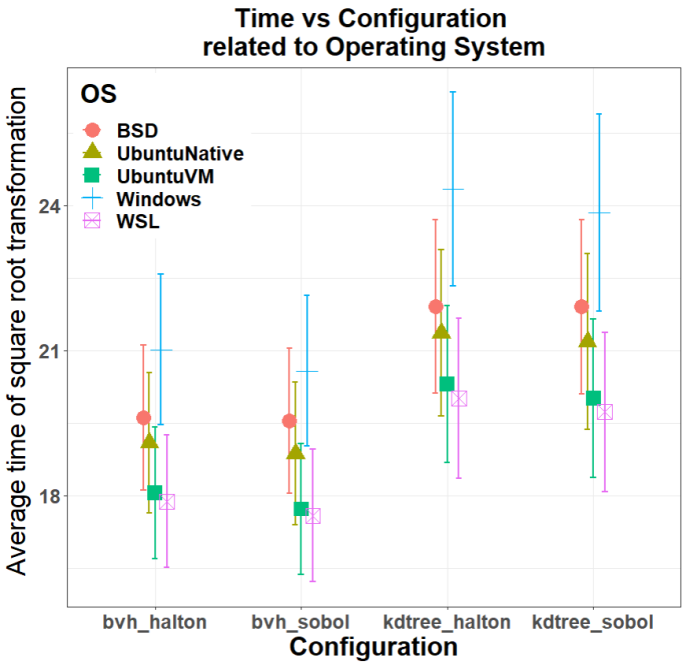

Fig. 24. Interaction between Operative System and configuration with data transformation

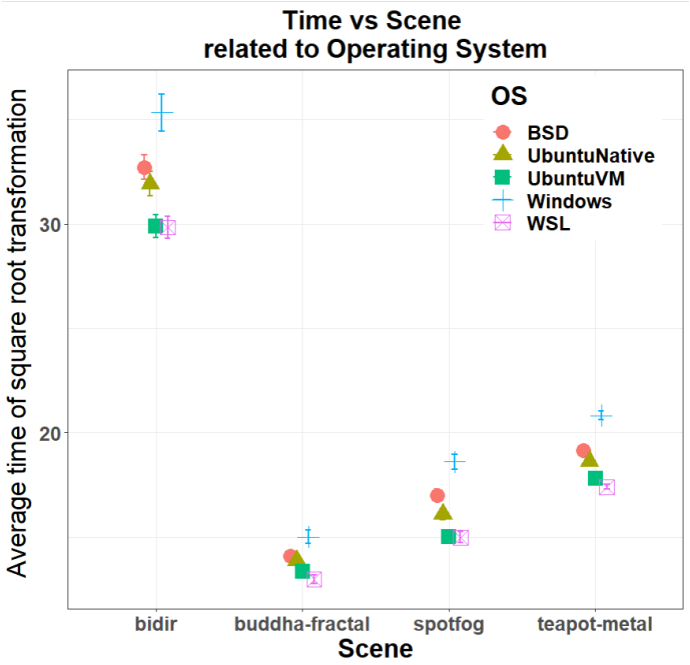

Fig. 25. Interaction between Operative System and scene with data transformation

If we apply a reverse transformation of our data to restore it to it's original state we can see the following results in figure 26, 27 and 28. Also it is important to analyze the original interactions in figure 29 and 30 to compare it against the transformed ones and we can notice that results are very similar and have similar tendencies.

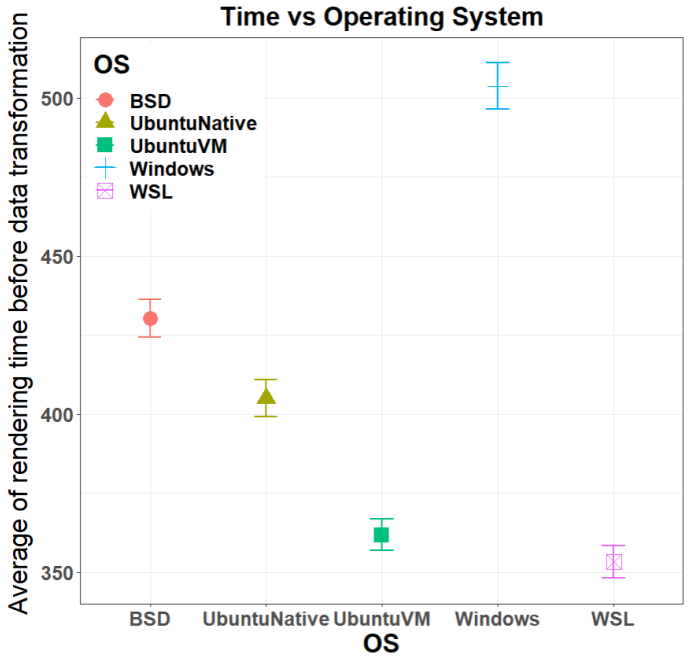

Fig. 26. Average of rendering time with original data for Operative System

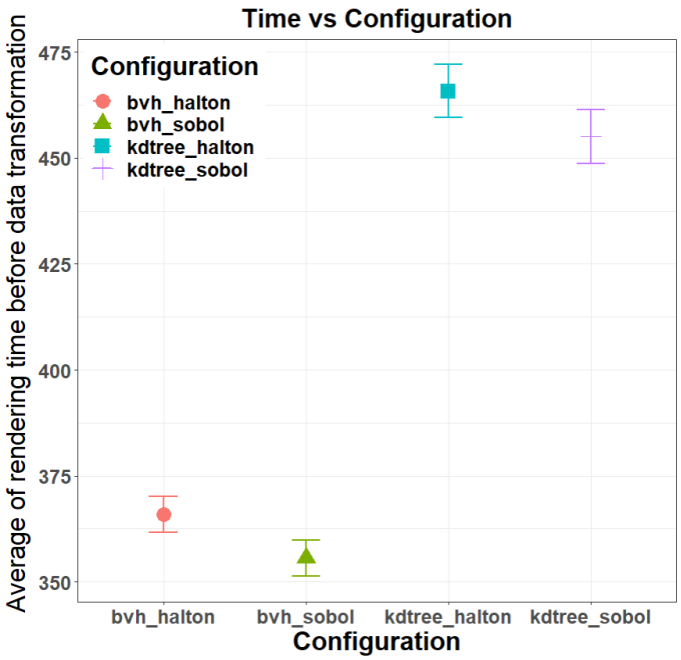

Fig. 27. Average of rendering time with original data for configuration

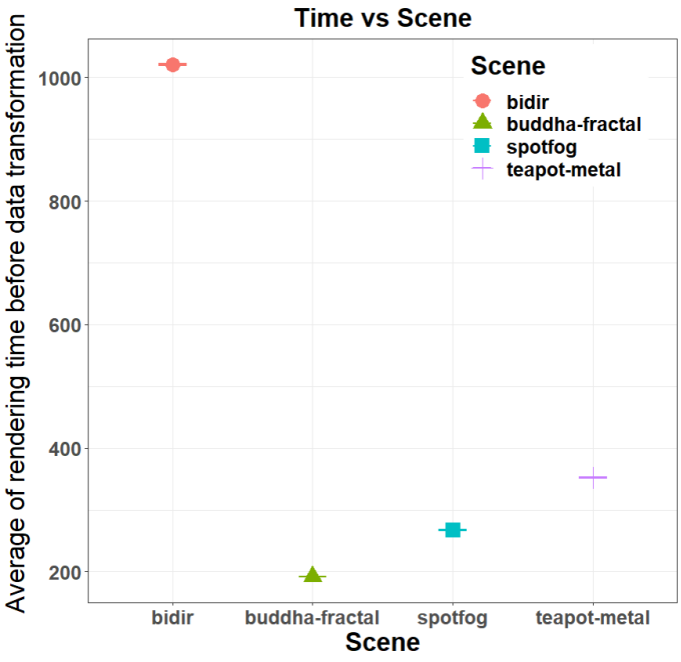

Fig. 28. Average of rendering timewith original data for scene

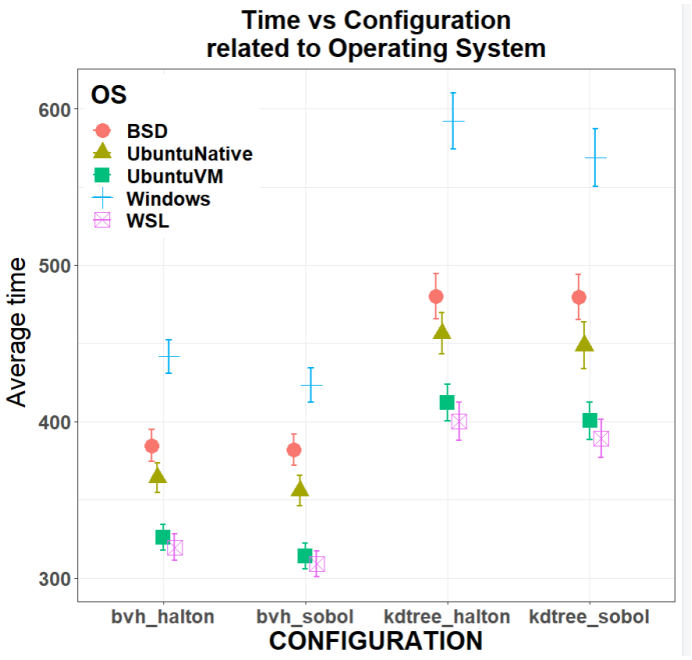

Fig. 29. Average of rendering time between operating system and configuration

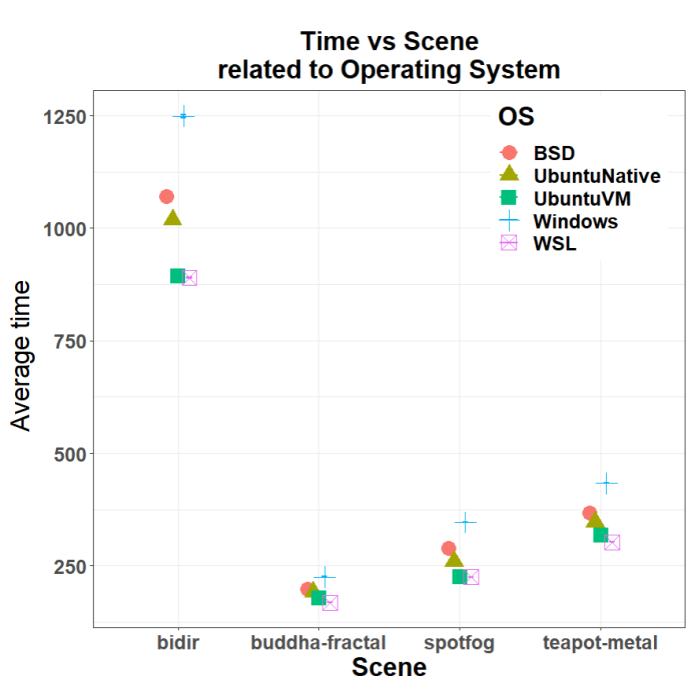

Fig. 30. Average of rendering time between Operative System and scene

As part of the final results, we present another interpretation of the original interactions differences using the Pairwise t-test (see tables IV, V andVI for individual factors and VII VIII for interactions pairs).

From the table IV, we can see that there is no statistically significant differences between WSL, Ubuntu Native, UbuntuVM and FreeBSD, which means that either of them are very comparable for our study. But when we compare Windows against WSL and UbuntuVM, these are statistically different. Which is something that is very interesting because both of these operating systems are installed on top of the Windows 11 machine, leading us to believe that WSL and VMWare (the program we used to install UbuntuVM) are very well optimized to have no statistically significant difference

between these and UbuntuNative and FreeBSD.

From the table V, we can notice that the configurations that involves **kdtree** are statistically different than **bvh** Also, in table VI we can notice that all scenes are significantly different because they were chosen on purpose to have different rendering times.

In the table IV, we can see that there are no significant differences between the Operating Systems and the Configurations. Meaning that the chosen rendering option parameters perform similar no matter the operating system.

Lastly, in the table VIII, we can see that for the $bidir$ scene, there are significant statistical differences between WSL, Ubuntu Native, FreeBSD and Windows, while WSL and UbuntuVM has no differences, this is the same behavior for the $spotfog$ and $teapotmetal$. And for $buddhafractal$ we noticed that there are no significant statistical differences between WSL, UbuntuVM and Ubuntu Native, being all from the same Linux distro, while it has significant statistical differences between WSL, FreeBSD and Windows.

	WSL	UbuntuVM	UbuntuNative	BSD
UbuntuVM	0.844548497			
UbuntuNative	0.352700499	0.41989154		
BSD	0.224213107	0.230443308	0.653788991	
Windows	0.014806917	0.014806917	0.144490038	0.230443308

TABLE IV
PAIRWISE T-TEST FOR OPERATING SYSTEMS

	bvh.halton	bvh.sobol	kdtree.halton
bvh-sobol	0.807238577		
kdtree-halton	0.033936232	0.033936232	
kdtree-sobol	0.047837576	0.033936232	0.807238577

TABLE V
PAIRWISE T-TEST FOR CONFIGURATION

	bidir	buddha.fractal	spotfog
buddha-fractal	3.27E-196		
spotfog	4.51E-174	1.19E-14	
teapot-metal	7.96E-150	5.05E-44	5.19E-14

TABLE VI
PAIRWISE T-TEST FOR SCENES

	WSL.bvh.halton	WSL.bvh.sobol	WSL.kdtree.halton	WSL.kdtree.sobol
UbuntuVM:bvh-halton	0.970651887			
UbuntuVM:bvh-sobol		0.972568574		
UbuntuVM:kdtree-halton			0.961527431	
UbuntuVM:kdtree-sobol				0.961527431
Ubuntu:bvh-halton	0.891593509			
Ubuntu:bvh-sobol		0.876645962		
Ubuntu:kdtree-halton			0.870954564	
Ubuntu:kdtree-sobol				0.862152147
BSD:bvh-halton	0.839151415			
BSD:bvh-sobol		0.805554526		
BSD:kdtree-halton			0.805554526	
BSD:kdtree-sobol				0.805554526
Windows:bvh-halton	0.702483029			
Windows:bvh-sobol		0.722263176		
Windows:kdtree-halton			0.566619515	
Windows:kdtree-sobol				0.566619515

TABLE VII
PAIRWISE T-TEST FOR OPERATING SYSTEMS AND CONFIGURATION. WE CHANGED THE LABEL OF UBUNTUNATIVE TO UBUNTU FOR THE SAKE OF THE TABLE TO FIT

	WSL.bidir	WSL.buddha.fractal	WSL.spotfog	WSL.teapot.metal
UbuntuVM:bidir	0.923709056			
UbuntuVM:buddha-fractal		0.470639933		
UbuntuVM:spotfog			0.985641012	
UbuntuVM:teapot-metal				0.466604692
Ubuntu:bidir	0.000175875			
Ubuntu:buddha-fractal		0.103672625		
Ubuntu:spotfog			0.046424403	
Ubuntu:teapot-metal				0.028246251
BSD:bidir	2.28E-07			
BSD:buddha-fractal		0.040507925		
BSD:spotfog			0.000258682	
BSD:teapot-metal				0.00159137
Windows:bidir	1.54E-21			
Windows:buddha-fractal		0.000216684		
Windows:spotfog			1.06E-10	
Windows:teapot-metal				1.03E-09

TABLE VIII
PAIRWISE T-TEST FOR OPERATING SYSTEMS AND SCENES. WE CHANGED THE LABEL OF UBUNTUNATIVE TO UBUNTU FOR THE SAKE OF THE TABLE TO FIT

V. CONCLUSION

With the presented results in the previous section, we can conclude that some oeprating systems are better at rendering physically based images than others. This could be related to optimizations for these types of tasks, but yet, we can not conclude that any operating system is better than other in the overall picture of systems functionality. Regarding our study topic we can conclude that Windows 11 is the worst performing of our study subjects for scenes and configurations of PBRT, and this difference is statistically significant. The best options for addressing these type of problems would be to use WSL or an Ubuntu VM, the average time of these systems was statistically better, and there is a really small difference between them. Other options like Ubuntu Native and FreeBSD are better than Windows but not as good as WSL, this might be related to optimization done on WSL. It is also very interesting that there are three clusters (visible in figure 29), where the first one is comprised of WSL and UbuntuVM (best performers), the second one would be FreeBSD and Ubuntu Native and lastly Windows as the third cluster (and also the worst performer).

For future work, we would like to evaluate another operating systems like YoctoLinux since its specialization could lead to

better results, also test Mac OS and experimental operating systems like MINIX3. We would also like to compare our results using PBRT-v4, which uses a dedicated GPU.

We have chosen to send this paper for review to the following conferences **csen2023** and **iscsi2023** in order to receive feedback and earn more experience with paper redaction in general.

APPENDIX

When trying to install $pbrt - v3$ on FreeBSD operating system, we encountered some issues that we had to troubleshoot and fix. These issues are related to the submodules that are imported by $pbrt$ that are called $ptex$ and $glog$. We have raised 2 Issues on GitHub, where the first one is on the $ptex$ repository here: https://github.com/wdas/ptex/issues/72 and along with the Issue, we proposed a solution in the following Pull Request: https://github.com/wdas/ptex/pull/73, which has been accepted and merged. As for the other issue related to the $glog$ project, we did not create any Issues because the version referenced by $pbrt - v3$ is too old. So we decided to create an Issue on the $pbrt - v3$ repository here: https://github.com/mmp/pbrt-v3/issues/327.

REFERENCES

[1] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, Sept. 2016. Google-Books-ID: iNMVBQAAQBAJ.

[2] D. Mould and P. Rosin, "Developing and applying a benchmark for evaluating image stylization," *Computers & Graphics*, vol. 67, June 2017.

[3] P. Hong, S. Hong, J. Roh, and K. Park, "Evolving benchmarking practices: A review for research perspectives," *Benchmarking: An International Journal*, vol. 19, pp. 444–462, July 2012.

[4] C. Bienia, "BENCHMARKING MODERN MULTIPROCESSORS,"

[5] J. Bradley, K. Chan, and D. Smith, "The Measured Performance of Personal Computer Operating Systems,"

[6] O. Örnvall, "Benchmarking Real-time Operating Systems for use in Radio Base Station applications,"

[7] Z. Wang, "The Development of Ray Tracing and Its Future,"

[8] P. H. Christensen, J. Fong, D. M. Laur, and D. Batali, "Ray Tracing for the Movie 'Cars'," in *2006 IEEE Symposium on Interactive Ray Tracing*, pp. 1–6, Sept. 2006.

[9] J. Schmittler, D. Pohl, T. Dahmen, C. Vogelgsang, and P. Slusallek, "Realtime Ray Tracing for Current and Future Games.," pp. 149–153, Jan. 2004.

[10] I. Wald, G. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navratil, "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, pp. 931–940, Jan. 2017. Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[11] A. Sheharyar and O. Bouhali, "A Framework for Creating a Distributed Rendering Environment on the Compute Clusters," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 6, 2013.

[12] N. Hatt and A. Sivitz, "Benchmarking Operating Systems,"

[13] D. C. Montgomery, *Design and analysis of experiments*. Hoboken, NJ: John Wiley & Sons, Inc, eighth edition ed., 2013.