Desktop Based Volunteer Computing Application to Perform Distributed Rendering

Jos Timanta Tarigan, Mahyuddin K. M. Nasution, Amer Sharif, Elviawati Muisa Zamzami Muhammad Ari Syahputra Sandan

Abstract*: The ability of computer to communicate to other computer gives a possibility to perform collaboration of multiple nodes to perform a complex computation. Distributed computing is one of the many solutions that use this idea. However, forming a set of computers dedicated to performing a single task may be too expensive. In this paper, we present a desktop based volunteered computing as an alternative to distributed computing. The system is able to perform distributed rendering on computer that voluntarily run the program. In the test, we use 5 computers with different specification connected to the server through the internet. By using this system, we were able to render a 3D scene collaboratively on 5 computers.

Index Terms: Volunteer Computing, Distributed Rendering, Computer Graphic.

I. INTRODUCTION

Using multiple hardware to solve a complex and massive computation is one of the most common solutions in computer industry. Task that can be split into smaller independent ones can be performed by multiple computers in parallel. This solution, which commonly known as grid computing or distributed computing, may significantly reduce processing time which may be crucial for some project. However, building a set of computers to perform a single task may be an expensive option, especially for a small to medium size company. Moreover, the usability of the resource can be low once there are no more tasks to be processed. Hence, building an infrastructure for grid computing is not a feasible option for small companies.

Volunteer computing is a concept where computing can be performed by other non-dedicated computers. In this architecture, users may voluntarily provide its computer to perform a task organized by a server. The advantage of volunteer computing system is it allows its user (or owner) to control the allocation of resources such as time and CPU usage.

In this paper, we propose a volunteer computing system that performs a rendering process of a 3d scene. The rendering process is done by using ray tracing algorithm. The algorithm calculates the color of each pixel based on the properties in the scene. Since the computations of each pixel are independent to each other the algorithm is highly parallel. Hence, distributed computing is an optimal solution to perform the rendering and may reduce the processing time significantly. Our objective is to build the proposed system, perform tests and collects the result. The proposed system is similar to our previous application [1] but with a different target platform. In this work, the volunteers are desktop computer instead of Android based mobile phone.

This paper is structured as follows: in the next section, we will go through researches that related to our work. Next, we will describe the architecture of our system. In the fourth part we will provide the result of our test. We will then conclude our work in the last section of this paper.

II. RELATED WORKS

The concept of ray tracing to render an image of 3-dimensional scene was first coined in 1980 by Whitted [2]. The idea of this concept is to track the path of rays from each point (or pixel) on the screen to the point in the scene. By tracking the lights bounced at the point, we can determine the color of the pixel. However, based on the amount of pixel on the screen and the complexity of the scene, the process may require a lengthy processing time.

To avoid the required time, computer graphic industry uses distributed rendering. The proposed solution performs rendering process in multiple computers. This solution is effective since pixel computation is independence to each other; a value on one pixel does not affect the other pixel. Research by Nonaka et al. [3], Kantert et al. [4], and Patil et al. [5] are a few examples of research in using multiple computers to render complex image. Cao et al. developed a distributed multi-GPU solution to perform parallel rendering [6]. Zotos et al. developed a distributed rendering application using Xbox 360 systems and personal computers as volunteers [7]. Liu et al. developed AzureRender, a method to perform real time rendering using a client server architecture [8]. The solution is proposed to be used in gaming where frame per second, which is important to keep animation smooth, is significant to user experience. By splitting the rendering process to client and server, the system may able to draw image at a higher frame rate.

Volunteer computing is a concept where users may voluntarily share their resources to complete a task collaboratively with other.

Revised Manuscript Received on April 25, 2019.

Jos Timanta Tarigan, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Indonesia.
Mahyuddin K. M. Nasution, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Indonesia.
Amer Sharif, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Indonesia.
Elviawaty Muisa Zamzami, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Indonesia.
Muhammad Ari Syahputra Sandan, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Indonesia.

One of the early projects involving volunteer computing is SETI@Home [9], a project to perform a search for extra-terrestrial intelligent (SETI) that requires massive processing power performed by Space Sciences Laboratory of the University of California, Berkeley. The project allows the use of large-scale distributed computing consist of volunteered users who willing to share their CPU cycles. Other interesting projects were done by both Castro et al. [10] and Kajitani et al. [11]. Castro et al. developed DENIS@Home, a system that perform collaborative simulation of electrophysiological models by using volunteered CPU, while Kajitani et al. uses volunteer computing to solve elliptic curve discrete logarithm problem. While most volunteer computing focus on native desktop application to achieve optimal solution, there are also works that focus on the ability of the system to be run on multiple platforms. Lavoie et al. developed Pando [12], a web-based volunteer computing solution that aim to ease prospective volunteer to join the program. The platform allows a volunteer computing project to be joined by simply running the web application via browser. Theodoropoulos et al. developed mCluster [13], a volunteer computing framework built for portable devices. The motivation of these projects is based on the fact that the number of mobile devices dwarfed the number of desktop computer. However, it is important to notice that volunteer computing performance does not only requires volunteers' processing power, but also data bandwidth, both speed and quota, to transfer the data. Monsalve et al. investigate the performance of volunteer computing for data intensive application [14]. The work investigates multiple volunteer computing systems and evaluates the data amount communicated during the process. It is important to keep the balance between CPU usage and bandwidth usage to keep the burden of volunteers on performing the task.

III. GENERAL ARCHITECTURE

The architecture of the system is similar to the distributed rendering from our previous work [1]. The system is based on the architecture of our previous project, Online Teamwork. But we are only using the server since the client is built on Android Platform.

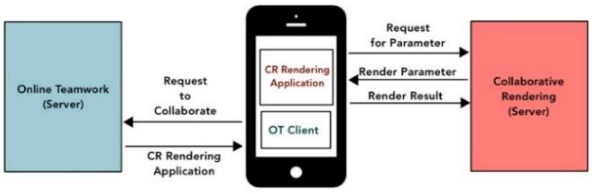

Fig. 1. Architecture of the System. Image courtesy of Tarigan et al.

Overall, the system consists of two parts: renderer and collaborator. The renderer is responsible for performing ray-tracing collaboratively across multiple volunteers using Online Teamwork platform. The renderer is split into two main parts: server and clients. Server is responsible for determining set of tasks to be distributed amongst volunteered client. These tasks, which will be stored in the database, will be picked by client by performing an HTTP request to request a task and send the result. The architecture of the program is illustrated in Fig. 1.

A. Client Application

Client application is basically a rendering program with a user interface. The rendering process uses basic ray-tracing algorithm which has the capability to render basic shading (ambient, diffuse, and specular) colors, soft shadow, full reflective, and refraction. Each pixel is multi-sampled to decrease the noise created by the algorithm. The ray-tracing program is built using basic Java desktop application. To increase the efficiency of CPU usage, we enhance the program to run as multiple processes instead of a single process based on the availability of cores on the CPU. Aside from the multi-threading optimization, there are no other optimizations made to the rendering process. While we are satisfied with the performance of our current rendering, there are many possible optimizations we can do in the future. It is also important to know that the ray tracing algorithm is modified to only calculate a single line of image instead of the whole image. Given a value n from the server, the client will calculate all pixels in row n in the image.

Another important factor in our application is the user may freely start and stop volunteering to the system. However, upon stopping the process, the calculated data will be lost. The current system does not have the capability to save part of an incomplete task.

B. Server Application

The server is responsible for distributing the task to clients. We built a simple web server to perform this task. The server uses Apache 2.4.18 and MySQL 5.6.39. All data is stored in the database by using the format defined below.

Table 1. Format on The Server's Database

Column	Type	Information
id	int	Primary key
row	int	row index of pixels to be rendered
result	text	Result in text formatted RGB values
accessed_by	Varchar	The last device accessed the task
accessed_time	Datetime	The last time the row was accessed
elapsed_time	Int	Elapsed time to finish the task in millisecond
status	Int	Status of the task: not accessed (0), unfinished (1), and finished (2)

Task management is an important factor of the system. There are many ways we can split our rendering problem into smaller tasks. However, it is important to consider that volunteers fully control their contribution to the process. Hence, it may increase the possibility of incomplete task. In

our current system, the server distributes the task to volunteers by following these set of rules:

- Each task contains all pixels in a row. Hence, if the resolution of the image is 1920x1080, one task consists of 1920 pixels.
- Every time a volunteer request for a task, the server will grab the first task with status not accessed. It will then set the status to unfinished (1) and update the accessed_time.
- An unfinished task expiry time is 24 hours; if any task is not finished after the expiry time, server will treat the task as an incomplete task and allow other users to recalculate the task.

To increase the usage of high-performance volunteers, we also allow an idle volunteer to process the unfinished task whether it is being processed by another volunteer or not. While this rule may decrease the efficiency (by allowing multiple volunteers to process a single task), it also may increase the rendering process.

IV. TEST AND RESULT

We performed the tests using 5 computers with specification listed in Table 2. The specification is varied from a quad core computer to dual core. During the test, all computers are connected to the same internet connection although not the same router. However, in the test, we let the users (who owns the computer) uses the computer as usual without concerning the progress of the computation on their computer. While this may increase the failed task, we decide that the factor of random failure is important to simulate the volunteer aspect of the system.

In this work, we focus our observation on two aspects: rendering time and CPU usage. We also collected elapsed time from each CPU to render a single task to evaluate the relation of system efficiency to volunteer's behavior. However, in the current test, we do not consider the main usage of the computer instead of performing the calculation. We are aware that some task (such as gaming, audio/visual activity) may significantly affect the allocation of CPU to the system and are planning to collect such data in the future.

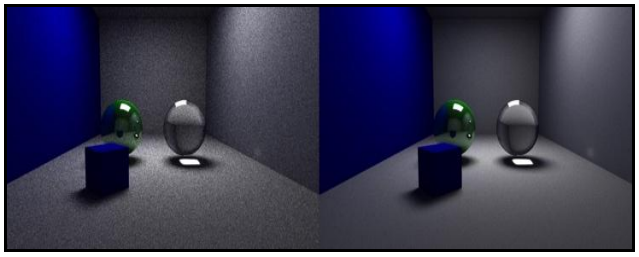

Fig. 2. Rendered Images with 256 samples (left) and 1024 samples (right) per pixel

Table 2. List of Volunteered Computing used in the test

Volunteer ID	Specification
cpu1	Intel i7-7700HQ @ 2.80GHz (4 CPUs), 16GB RAM, Windows 10 Home 64-bit
cpu2	AMD A4-4000 @ 3.00Ghz (2 CPUs), 6 GB RAM, Windows 10 Home 64-bit
cpu3	Intel i5 1.6 GHz (2 CPUs), 4 GB RAM OS X 10.11El Capitan
cpu4	Intel i7 @ 2.3 GHz (4 CPUs), 16GB RAM, macOS High Sierra
cpu5	Intel i5-3230 @ 2.60GHz (4 CPUs), 4GB RAM, Windows 10 Home 64 bit

The rendering target is a basic Cornell box scene as our sample which contains 5 planes as walls, 1 diffuse coloured box, 1 reflective sphere, and 1 refractive sphere. We perform the rendering twice; one with 256 samples per pixel and another one with 1024 samples per pixel. The result image is shown in Fig. 2.

The table below shows the average rendering time for each task from each volunteer. It is important to remember that pixel processing time is highly depending on the complexity of the scene on that point. Reflective and refractive materials are harder to compute than diffuse material. Hence, each task may require more time to complete those others. It is also important to know that the average rendering time does not include the time needed to transfer the data to the server. We assume that data transfer time is not significant since in our case the maximum character contained in each task is 60.000 characters (~23KB/task).

As expected, cpu1 has the highest performance amongst all others followed by cpu4. Cpu1 was able to complete 30 percent of the task on the first test. Cpu2, which has the lowest specification, could only complete 10% of the task on both tests. Overall, the system able to finish the rendering process in 42 minutes for 256 samples per pixel image.

Table 3. Test Result for 256 Samples per Pixel

Volunteer ID	Task Performed (Amount/Percent)	Average Time Per Task (Second)
cpu1	321 / 30	2.5
cpu2	112 / 10	6.2
cpu3	151 / 13	7.0
cpu4	270 / 25	3.1
cpu5	226 / 20	4.1

In the second test, we modify the renderer to perform 1024 samples per pixel. Table 4 below shows the result of the test. As expected, the result shows a similar trend with the first test with cpu1 were able to compute 32% of all the tasks followed by cpu4 and cpu5. On average, completing the task in the second task took almost 4 times longer than the first test.

Another interesting fact is that increasing the sample, which makes the task heavier, may increase the chance of failed task. In the second test, 13.1% of the task were computed twice or more due to the failure of the first volunteer to finish the task. This failure rate is an increase from the first test (8.1%). This is expectable due to longer task may increase the chance of incomplete calculation.

Table 4. Test Result for 1024 Samples per Pixel

Volunteer ID	Task Performed (Amount/Percent)	Average Time Per Task (Second)
cpu1	341 / 32	8.8
cpu2	111 / 10	22.0
cpu3	161 / 15	14.9
cpu4	282 / 26	10.3
cpu5	185 / 17	14.1

V. CONCLUSION AND FUTURE WORKS

In this research, we have successfully performed 3D rendering in volunteer computing platform. The test is done by using 5 volunteers computing connected to a server through the internet. This solution can be used in small to medium company to use their idle computers to help renderer to perform the rendering. This solution allows users to keep using their computer while volunteering some of their processing power to perform the rendering in multiple devices. We are confident that given a set of volunteers to perform the rendering process, using our system can significantly decrease the amount of time needed to create a 3D content. However, this is highly depends on the behavior of the volunteers.

One of the significant problems raised during the test is the amount of incomplete task wasted when the user chose to stop volunteering. The current system is unable to save the computed pixel. Since the system allowed user to actively start and stop volunteering, there will be a high amount of wasted computation during the process. While the test does not show this problem, our objective is to allow the system to be installed in personal computers. Users may repeatedly start and stop the process and may increase inefficiency. The future research will focus on solving this problem.

Another significant feature we would like to add in the system is to allow the volunteers to pick which task they want to perform. This feature is already available in our previous project [1] but has yet to arrive in the desktop counterpart.

ACKNOWLEDGEMENT

The authors would like to thank the head and staff of "Lembaga Penelitian USU (Research Center of Universitas Sumatera Utara)", The author would also like to thank the Dean of Faculty of Computer Science and Information Technology University of Sumatera Utara, Prof. Dr. Opim Salim Sitompul, M.Sc for their widest to support this work.

REFERENCES

1. J. T. Tarigan, I. Jaya, and S. M. Hardi, "Distributed Rendering on Volunteered Mobile Resources," In *Proceedings of the 5th International Conference on Communications and Broadband Networking*. pp. 25–29, 2017.
2. T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.
3. J. Nonaka, N. Sakamoto, T. Shimizu, M. Fujita, K. Ono, and K. Koyamada, "Distributed Particle-Based Rendering Framework for Large Data Visualization on HPC Environments," *International Conference on High Performance Computing & Simulation*, pp. 300–307, 2017
4. J. Kantert, H. Spiegelberg, S. Tomforde, J. Hahner, and C. Muller-Schloer, "Distributed Rendering in an Open Self-Organised Trusted Desktop Grid," *EEE International Conference on Autonomic Computing*, pp. 267–272, 2015.
5. G. V. Patil and S. L. Deshpande, "Distributed rendering system for 3D animations with Blender," *IEEE International Conference on Advances in Electronics, Communication and Computer Technology*, pp. 91–98, 2016.
6. Y. Cao, H. Wang, and Z. Ai, "Distributed Multi-GPU Accelerated Hybrid Parallel Rendering for Massively Parallel Environment," *International Conference on Virtual Reality and Visualization*, pp. 30–36, 2014.
7. E. Zotos and R. Herpers, "Interactive Distributed Rendering of 3D Scenes on Multiple Xbox 360 Systems and Personal Computers," *International Conference on Cyberworlds*, pp. 114–121, 2012.
8. Z. Liu and H. Zou, "AzureRender: A Cloud-Based Parallel and Distributed Rendering System," *IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 1881–1886, 2015.
9. E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "SETI@home-massively distributed computing for SETI," *Computing in Science & Engineering*, vol. 3, no. 1, pp. 78–83, 2001.
10. J. Castro, V. Monasterio, and J. Carro, "Volunteer Computing Approach for the Collaborative Simulation of Electrophysiological Models," *IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 118–123, 2016.
11. S. Kajitani, Y. Nogami, S. Miyoshi, and T. Austin, "Volunteer Computing for Solving an Elliptic Curve Discrete Logarithm Problem," *Third International Symposium on Computing and Networking* , pp. 122–126, 2015.
12. E. Lavoie, L. Hendren, and F. Desprez, "Pando: A Volunteer Computing Platform for the Web," In *Proceedings of the 5th International Conference on Communications and Broadband Networking*, pp. 387–388, 2017.