UDC 004.92/.94

Y. Danyliuk, O. Zhurakovska

EFFECT OF RENDERING SPACE ONTHE PERFORMANCE OF THE RAY MARCHING METHOD

Abstract: This study investigates the efficiency of rendering implicit surfaces using two distinct approaches: screen rendering space and bounding volumes rendering space.

Through comprehensive analysis, visualization of rendering differences, and performance testing, we evaluate the effectiveness of each approach. Our findings demonstrate significant advantages offered by bounding volumes rendering space, including efficient elimination of empty areas, reduction of ray marching iterations, and enhanced CPU and GPU resource utilization. Notably, bounding volumes rendering space achieves an 87% increase in average framerate compared to screen rendering space, achieving 102.71 FPS versus 54.82 FPS, respectively.

Our results underscore the promise of bounding volumes rendering space as a potent approach for improving rendering efficiency in ray marching-based implicit surface visualization. We advocate for its integration into rendering pipelines and suggest avenues for future research to maximize its benefits and impact on computer graphics and visualization.

Keywords: bounding volumes, implicit surfaces, performance testing, ray marching, rendering efficiency, rendering space, SDF, Unity

Introduction

Actuality

In the realm of scientific computing and graphics, implicit surfaces play a crucial role in interpreting discrete data, serving as a fundamental tool for data analysis across various disciplines [1]. These surfaces provide a flexible way to depict intricate phenomena, especially when conventional visualization techniques fall short.

Implicit surfaces are highly valued for their ability to represent intricate shapes with minimal data, making them widely utilized across various fields, including modeling, animation, physics, and mathematics simulations. With the propagation of 3D scanning technologies, such as computed tomography (CT) and magnetic resonance imaging (MRI), the demand for precise and efficient visualization methods has intensified [2].

Problem

Implicit surfaces present a challenging and resource-intensive task in rendering and visualization. Key to this challenge is refining the rendering process to achieve both accuracy and efficiency. In addressing this, researchers have explored various methods,

 ISSN 1560-8956

including three main approaches: extracting explicit polygonal geometry, resampling the implicit into proxy geometry, or directly ray tracing or marching the implicit [1]. While these methods offer distinct advantages, ray marching stands out for its potential to provide accurate representations directly from implicit data.

Although ray marching offers a promising option, it can be computationally intensive, necessitating exploration to maximize its efficiency. One underappreciated solution, bounding volumes, shows potential in this regard. Despite being less popular, it offers a promising avenue for improving efficiency, as we aim to demonstrate in this study through comparative analysis. Evaluating the performance of this rendering technique is essential for advancing scientific computing and graphics, enabling researchers to identify the most suitable approach for their specific application domains.

Related research

In the realm of rendering implicit surfaces with ray marching, the technique of bounding volumes emerges as a potential solution to enhance performance [3]. Despite its proven effectiveness, bounding volumes remain underutilized, as evidenced by the lack of integration into various game frameworks, for instance, such as the "Raymarching Toolkit for Unity" by Kevin Watters and Fernando Ramallo [4]. While efforts have been directed towards improving ray marching performance by reducing the number of steps [5, 6], bounding volumes have been overlooked, likely due to their low popularity.

John Hart's work on sphere tracing highlights the utility of bounding volumes in culling processing of intricate geometries irrelevant to the current task [3]. However, despite their benefits, the technique has not received sufficient attention in the quest for rendering efficiency. In the pursuit of performance enhancements, researchers have explored various strategies, including over-relaxation-based acceleration methods and dynamic prevention of self-intersections [5]. These advancements have led to significant improvements in rendering quality and performance, particularly in real-time rendering scenarios.

In the context of virtual reality terrain rendering using ray marching, enhancement techniques such as distance hints have been investigated to reduce the number of ray marching steps per pixel [6]. While these techniques show promise, their effectiveness can vary depending on the specific application and hardware setup. Despite the advancements in improving ray marching, the potential of bounding volumes to significantly boost performance has been largely overlooked.

In light of these observations, this study aims to underscore the importance of bounding volumes as a potent approach for enhancing the efficiency of ray marching in rendering implicit surfaces. Through comparative analysis, we seek to demonstrate the considerable performance benefits that bounding volumes can offer in various scenarios. By

bringing attention to this underappreciated technique, we hope to encourage researchers to explore its potential and integrate it into their rendering pipelines for improved efficiency and quality.

Rendering approaches comparison and explanation

In our study, we built upon the foundation laid out in previous work [7]. Shortly saying, this system, developed in Unity utilizing fragment shaders, employs traditional sphere tracing with draw distance checks. In our implementation, we utilize specialized components within the Unity scene hierarchy to represent ray marched objects and create a representation of the virtual ray marching scene within the shader, a technique similar to that used in Kevin Watters and Fernando Ramallo's framework [4].

We designed two separate Unity scenes to illustrate the differences between screen space rendering and bounding volumes. As depicted in Fig. 1, in the first scene, we employed screen rendering space, where a quad is positioned just in front of the camera, and ray marching is applied for each pixel of the screen. Using this approach, we essentially render the entire screen space, encompassing potentially empty areas.

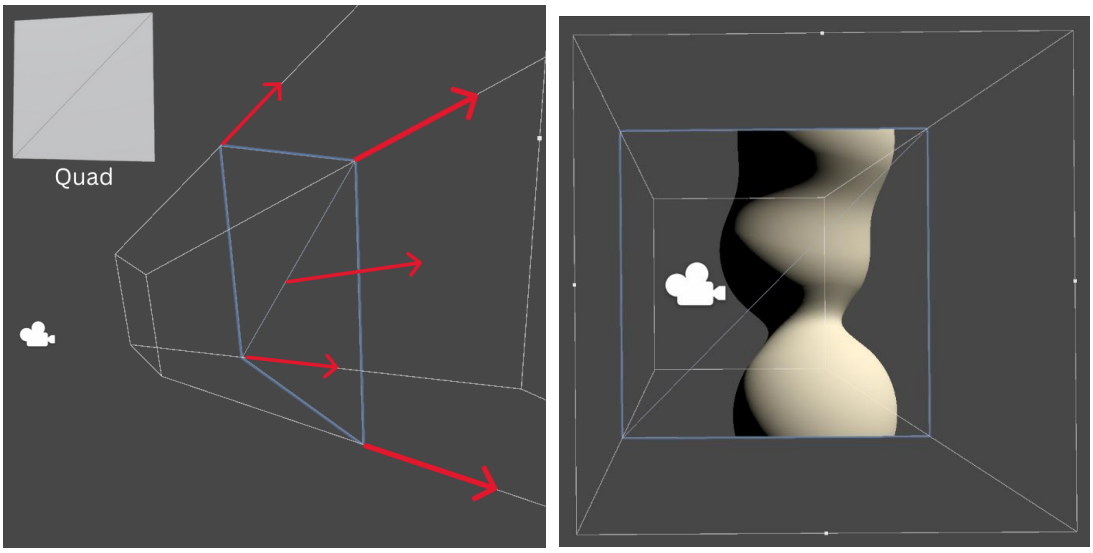

Figure 1. Process of ray marching using screen rendering space
(left: ray marching in rendering space, right: technical view of rendering result)

Conversely, in the second scene, each object was assigned its own rendering space, that can be shaped according to specific scenarios. As it’s showed in Fig. 2, we opted to use a Pentakis Dodecahedron, due to its efficiency in defining an accurate sphere shape with minimal vertices and triangles. This choice allowed for effective utilization of rendering space and facilitated operations such as smooth blending between objects, ultimately reducing the overall pixel count required for ray marching.

 ISSN 1560-8956

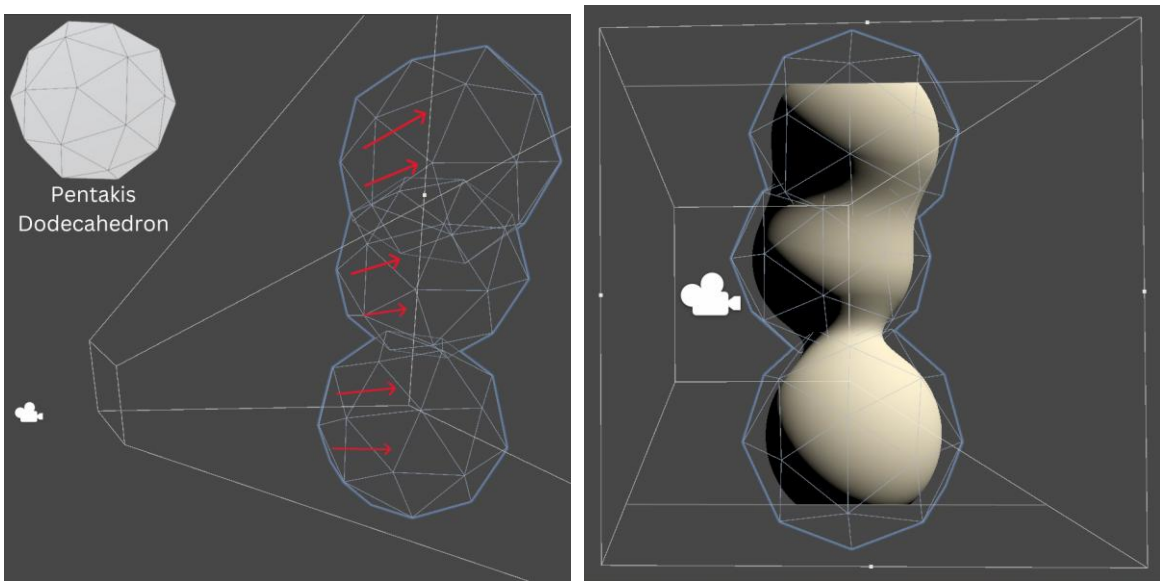

Figure 2. Process of ray marching using bounding volumes rendering space
(left: ray marching in rendering space, right: technical view of rendering result)

In these visual representations, red arrows indicate the direction of rays in sphere tracing, with the light gray representing the camera frustum and meshes polygons just like it displaying in the Unity editor. Notably, the light blue highlighted areas denote the rendering area, with the bounding volumes scenario showcasing a significantly smaller overall rendering area. This reduction in rendering area is visually apparent and highlights the efficiency gained through the utilization of bounding volumes.

Experimental setup and scene complexity design

In preparation for measuring rendering performance, our first step is to construct a more complex scene with a substantial number of objects. This serves to distribute the computational load across the physical resources of the computer, ensuring a robust testing environment.

Our experiment comprises two distinct Unity scenes, each featuring a collection of 30 simple 3D sphere Signed Distance Functions (SDFs). These spheres are organized into groups of three, with smooth blending SDF Boolean operations applied within each group, utilizing Constructive Solid Geometry (CSG) techniques. This arrangement creates sample objects for our experiment, with the spheres positioned closely together within each group to ensure visible blending effects.

In Fig. 3 you can observe rendering output, we make up the scenes such a way that for both approaches it remains visually the same.

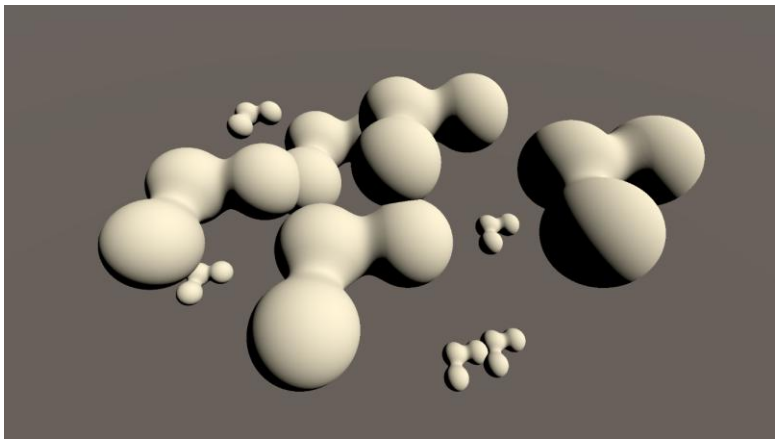

Figure 3. Experiment scene (same rendering output for both approaches)

The primary composition consists of five groups positioned near the camera, covering a significant portion of the screen. This deliberate setup aims to showcase the performance boost achieved, even in scenarios where bounding volumes may be less effective due to limited empty space. Conversely, the remaining five groups are positioned much farther from the camera, illustrating the performance differences in rendering for such scenarios.

Visualization and analysis of rendering differencesTop of Form

To assess the rendering differences between our scenes, we delve into the internal workings of the ray marching algorithm. Due to the challenges of debugging shaders, we rely on heatmaps to effectively visualize the number of ray marching iterations spent in calculations for each individual pixel.

In Fig. 4, we present a heatmap of our experiment scene utilizing screen rendering space.

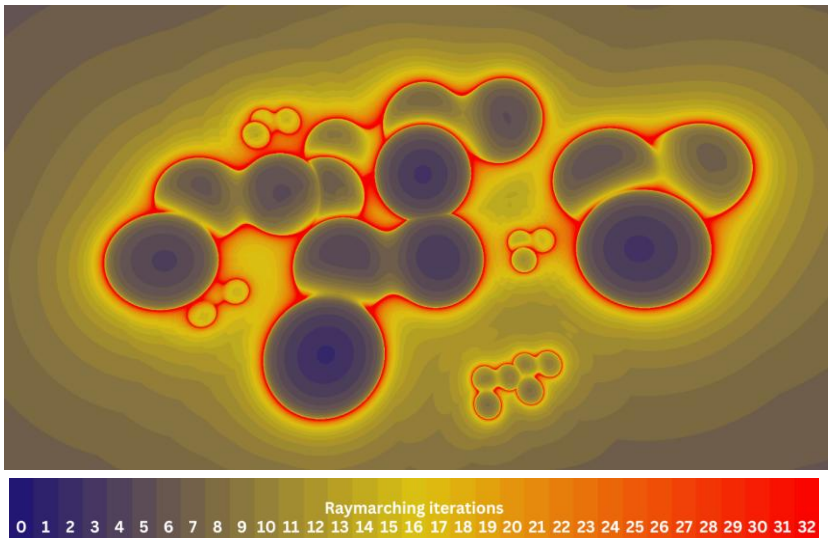

Figure 4. Experiment scene heatmap (screen rendering space)

 ISSN 1560-8956

For comparison, Fig. 5 displays a heatmap of the same scene using bounding volumes rendering space.

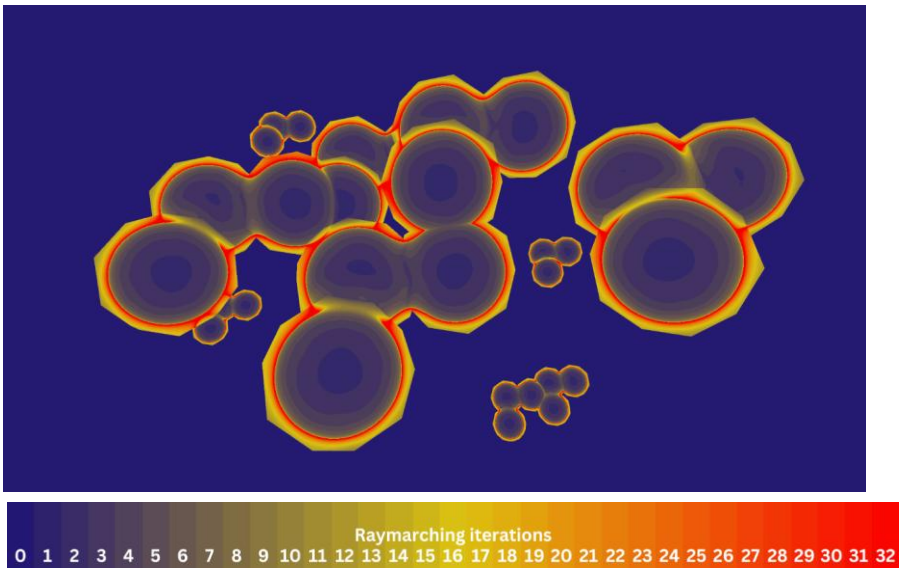

Figure 5. Experiment scene heatmap (bounding volumes rendering space)

A notable observation is the efficient elimination of empty areas in the second heatmap, which occupy approximately 67% of the entire screen area. This accounts for about 1,390,500 individual pixels, a significant portion for the sensitive ray marching algorithm. As we will demonstrate later, this refinement leads to a substantial performance boost.

Additionally, a less conspicuous yet significant improvement is the uniformity in the number of iterations required for distant objects compared to those closer to the camera. Unlike screen space rendering, where rays originate from the camera to all objects, bounding volumes allow for individual ray origins close to each object. Consequently, distant objects require the same number of iterations as those nearby, contributing to the overall performance enhancement provided by bounding volumes.

To gain further insights into the rendering process, we analysed the distribution of ray marching iterations for the first 10 largest screen areas covered by the scene. Table 1 presents the results of this analysis for both approaches, respectively.

In the case of screen rendering space, the distribution reveals that the majority of the screen area is rendered with iterations ranging from 7 to 11. The highest percentage of screen area, 15.4%, is rendered with 8 iterations.

In contrast, employing bounding volumes significantly reduces the range of iterations, with a noteworthy portion of the screen area rendered with 0 iterations. This indicates that a substantial portion of the screen is efficiently rendered without the need for any iterations. For the actual rendering part within objects, the range of iterations is limited to 1 to 6, as distant objects undergo similar iterations as close ones. Ignoring empty areas, the highest percentage of screen area is rendered with 3 iterations, accounting for 7.8%.

Table 1

Experiment scene ray marching iterations distribution

Screen rendering space										
Screen area (%)	15.4	13.8	10.2	9.7	7.3	5.5	3.6	3.5	3.1	2.7
Iterations	8	9	10	7	11	12	13	6	5	14
Bounding volumes rendering space										
Screen area (%)	67.1	7.8	5.7	2.7	2.2	1.3	0.7	0.6	0.5	0.5
Iterations	0	3	4	5	1	6	7	15	32	17

While the difference may seem insignificant at first glance, as scene complexity scales, the disparity in iteration usage for different areas becomes more pronounced. Moreover, through experimentation, we found that employing 36 iterations as the limiter for the bounding volumes scene is sufficient to render the scene without any visual errors. In contrast, the scene utilizing screen space as the rendering space requires 64 iterations as the limiter. This highlights a significant efficiency gain, as fewer iterations are needed for bounding volumes rendering compared to screen space rendering, with some pixels requiring up to twice as many iterations in the screen space scenario.

Comparative rendering performance analysis

To quantify the difference between the two approaches, we conducted performance testing to measure the rendering efficiency of both scenes and facilitate comparison. For this purpose, we employed the Performance Testing Extension package for Unity Test Runner framework [8] to gather rendering efficiency metrics. Additionally, we developed a custom solution using the mplsoccer Python library [9] to create a visually informative radar chart for comparative analysis.

The performance testing was conducted on a laptop equipped with an Intel(R) Core (TM) i7-10750H CPU 2.60GHz with 6 cores and 12 logical processors and an NVIDIA GeForce RTX 2060 GPU with 6.0 GB of dedicated GPU memory. These specifications ensured a robust testing environment capable of capturing rendering performance accurately.

In selecting performance indicators, we prioritized those that provided the most descriptive insights into rendering efficiency. These indicators were carefully chosen to provide a comprehensive understanding of performance differences between the two rendering approaches. The rendering performance radar chart, presented in Figure 6, depicts the results of rendering our experiment scene over a duration of 60 seconds. This duration ensures high precision and stability of the average values calculated, facilitating a comprehensive comparison between screen and bounding volumes rendering spaces.

 ISSN 1560-8956

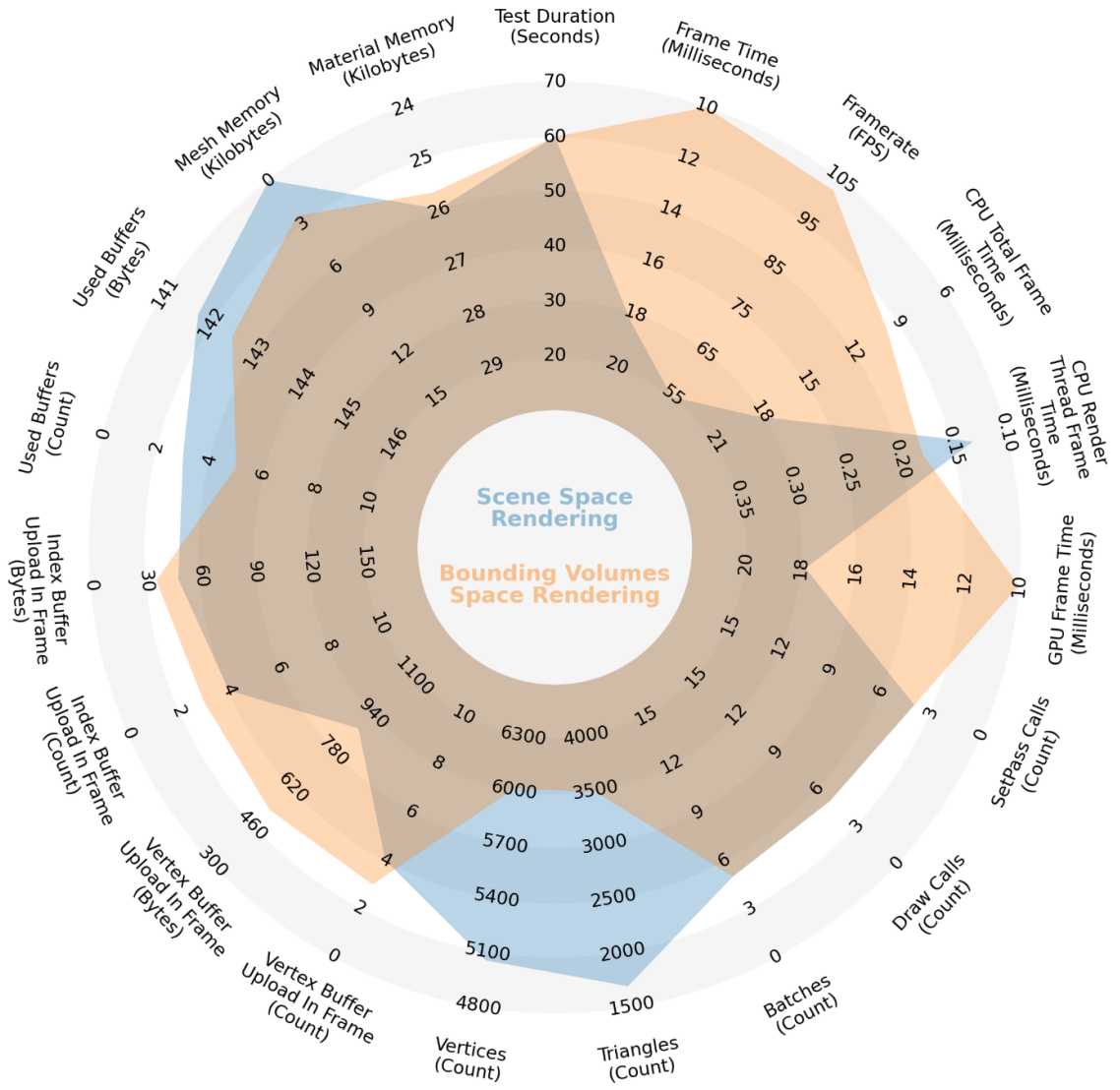

Figure 6. Rendering performance comparison radar chart
(average values calculated from data measured during 60 seconds)

As we've presented only average values, it's essential to note their reliability. The distributions for both approaches demonstrate dispersion within relatively small ranges centred around the average value. Given the stability ensured by the testing interval, we can confidently assert that the average values accurately depict the performance landscape.

Comparing the frame time between the two rendering approaches reveals a significant difference. The frame time for bounding volumes space rendering averages approximately 47% lower than that of the screen rendering space approach, clocking in at around 9.76 milliseconds compared to 18.35 milliseconds, respectively. This reduction translates to a remarkable 87% increase in the average framerate for bounding volumes space rendering, achieving 102.71 FPS on average, compared to the 54.82 FPS of screen

rendering space. Such a substantial improvement, roughly twice as high, suggests significantly smoother rendering and potentially a far superior user experience with the bounding volumes approach.

Furthermore, both CPU and GPU frame times witness notable reductions for the bounding volumes space rendering scene. This indicates that this technique utilizes both CPU and GPU resources more efficiently than the screen rendering space approach, contributing to an overall improved performance.

In summary, the performance testing reveals that the bounding volumes space rendering approach surpasses the screen rendering space approach in various key metrics. Although it involves a higher count of vertices and triangles due to utilizing more meshes, this factor has a minimal impact on performance compared to the more substantial gains in framerate and frame time reductions. Therefore, it's evident that the bounding volumes space rendering approach emerges as the more effective choice for enhancing performance in ray marching methods.

Conclusions

In this study, we conducted a thorough examination of rendering implicit surfaces using two distinct approaches: screen rendering space and bounding volumes rendering space. Through an in-depth analysis of the rendering process, visualization of rendering differences, and performance testing, we aimed to provide valuable insights into the efficiency of these approaches.

Our investigation revealed significant advantages offered by the bounding volumes rendering space approach over traditional screen rendering space. By leveraging bounding volumes, we were able to efficiently eliminate empty areas, reduce the number of ray marching iterations, and enhance CPU and GPU resource utilization. All this together culminated in a substantial improvement in rendering performance, with bounding volumes rendering space achieving a remarkable 87% increase in average framerate compared to screen rendering space.

Furthermore, our experiments demonstrated the scalability and adaptability of bounding volumes rendering space, showcasing its effectiveness across various scene complexities. Even in scenarios where empty space is limited, bounding volumes proved to be a superior choice, offering smoother rendering and potentially enhancing the user experience.

In conclusion, the results of our comparative rendering performance analysis highlight the promise of bounding volumes rendering space as a potent approach for improving rendering efficiency in ray marching-based implicit surface visualization. We advocate for greater attention to this underappreciated technique and its integration into rendering pipelines to unlock its full potential. As the demand for precise and efficient

 ISSN 1560-8956

visualization methods continues to grow, the insights gleaned from this study offer valuable guidance for future research and development endeavours in the field.

Moving forward, further research and development in this area could explore additional refinements to maximize the benefits of bounding volumes space rendering. Additionally, investigating the applicability of this approach with other rendering techniques and different scenarios could provide valuable insights into its broader impact on computer graphics and visualization.

REFERENCES

1. *Knoll, A.* A Survey of Implicit Surface Rendering Methods, and a Proposal for a Common Sampling Framework / Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI). – 2007. – P. 164-177.

2. *De-Araújo, B.R., Lopes, D., Jepp, P., Jorge, J., & Wyvill, B.* A Survey on Implicit Surface Polygonization / ACM Computing Surveys. – 2015. – Vol. 47. – P. 1-39. – DOI: 10.1145/2732197.

3. *Hart, J.* Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces / The Visual Computer. – 1995. – Vol. 12. – DOI: 10.1007/s003710050084.

4. *Watters, K., & Ramallo, F.* Raymarching toolkit for unity: a highly interactive unity toolkit for constructing signed distance fields visually / ACM SIGGRAPH 2018 Studio (SIGGRAPH '18). – Association for Computing Machinery, New York, NY, USA, Article 9. – 2018. – P. 1–2. – DOI: 10.1145/3214822.3214828.

5. *Keinert, B., Schäfer, H., Korndörfer, J., Ganse, U., & Stamminger, M.* Enhanced Sphere Tracing / Smart Tools and Applications in Graphics. – 2014.

6. *Brown, M. R.* Performance Techniques for Rendering Procedurally Generated Terrain in Virtual Reality Using Ray Marching / Bachelor of Science thesis, New Mexico State University, Department of Computer Science, Las Cruces, New Mexico. – 2017.

7. *Danyliuk, Y., Zhurakovska, O.* Information system for visualizing four-dimensional objects based on the ray marching method / V International Scientific and Practical Conference of Young Scientists and Students “Software Engineering and Advanced Information Technologies (SoftTech-2023)”. – December 2023.

8. *Unity Technologies* Performance Testing Extension. – 2019.

9. *Durgapal, A., & Rowlinson, A.* mplsoccer: Football pitch plotting library for Matplotlib / Retrieved from: https://github.com/andrewRowlinson/mplsoccer. – 2021.