Enhancing Rendering Performance in Complex Visualizations by using Optimization Techniques and Algorithms in Browser Environments

Sanja Brekalo
Medimurje University of Applied Sciences in Cakovec, Croatia
sbrekalo@mev.hr (corresponding author)

Klaudio Pap
Faculty of Graphic Arts, University of Zagreb, Croatia
klaudio.pap@grf.hr

Bruno Trstenjak
Medimurje University of Applied Sciences in Cakovec, Croatia
btrstenjak@mev.hr

Received: 6 March 2024 | Revised: 22 March 2024 and 25 March 2024 | Accepted: 27 March 2024

 DOI: https://doi.org/10.48084/etasr.7201

ABSTRACT

This research is based on the hypothesis that optimization techniques can significantly improve the performance of complex visualizations in web browsers. The aim of the former was to determine to which extent the optimization can be achieved. Optimizations were coded to improve visualization, reduce the need for visual rendering, and decrease script execution time as well as the needed resources. To test the hypothesis, various optimization methods and algorithms were implemented on the initial visualization script and were tested. The main goal of this implementation was to assess how optimization methods, including quadtrees, spatial hashing, binning, LOD adjustments, and the use of the map data structure, affect the performance of web visualization. The obtained results confirmed the hypothesis and the original animation was significantly improved. The implementation of optimizations had a positive effect on the performance of visualizations. The conducted tests gave concrete evidence confirming the validity of the initial hypothesis. This led to certain conclusions regarding which methods provide the best results when optimizing complex visualizations. Key recommendations for code optimization, which can be used in the development of complex visualizations in web browsers, were derived.

Keywords-optimization; complex visualizations; browser environments; spatial indexing algorithms; performance enhancement

I. INTRODUCTION

The challenge of effectively rendering complex visualizations is a significant area of research in the field of computer graphics and data visualization. The demand for more complex visual presentations is growing, and thus the need for optimization techniques to improve visualizations. Optimization algorithms have been utilized in diverse scientific and industrial domains to improve efficiency. Various optimization techniques and algorithms have been applied to enhance the effectiveness in processing and displaying data to varying degrees of success [1, 2]. The application of optimization techniques offers the potential of increasing the efficiency of displaying complex visualizations in real time. Advances in this area highlight the critical role of optimization in overcoming problems associated with computational limitations and performance deficiencies. Techniques, such as quadtrees [3], spatial hashing [4], binning, and Level-of-Detail (LOD) adjustments [5], along with code improvements using data structures like Map were chosen to be tested as effective strategies for optimizing rendering. The importance of optimization in rendering is not a new concept, however, its application within browser environments by deploying p5.js for complex visualizations, is a relatively unexplored area. This paper aims to demonstrate how a combination of optimization techniques can ameliorate rendering performance in such scenarios. Building on the work of implementing optimization with algorithms in various fields [6-9], this study extends those

principles to the specific challenges associated with rendering complex visualizations in browser environments.

JavaScript library p5.js was chosen for testing purposes. It is a library developed for learning, prototyping, electronic art, visualizations, visual programming and creating interactive installations [10]. p5.js is used in web environments, utilizing the JavaScript language. In addition to serving as a tool for artistic expression and design, p5.js is employed in education to introduce programming and computer graphic concepts to students in an intuitive way enabling those who are at introductory programming levels to master a wide range of academic concepts more effectively [11].

Despite the extended research in the field of rendering performance optimization, this paper recognizes and explores a specific niche within the p5.js environment. p5.js is a tool widely used in education and digital art, but less explored in the context of advanced performance optimization. This research attempts to fill this gap through detailed analysis and application of various optimization techniques, thereby providing new insights that are directly applicable and useful to the developer and research community.

II. VISUALISATIONS DEVELOPED FOR TESTING

HTML documents with p5.js script were created for the purposes of testing. The constructed animation was a flow field animation that utilizes the concepts of vector fields and Perlin noise [12] to generate a dynamic animation. The developed animation simulates natural flows [13] through the interaction of particles with a generated flow field. The flow of particles can simulate natural phenomena, such as wind, water, or flock behavior, and is applied to achieve more natural animations. The flow field in the code is made from vectors with a specific direction representing the direction of particle movement. The former changes in each drawing cycle by altering the Perlin noise, resulting in a dynamic animation which is transformed over time. This script was selected for testing due to its capability to be evaluated across various degrees of complexity. Complexity can be easily introduced and amplified by adding more particles to the animation. Firstly, the original animation was created, and then it was optimized in multiple ways with various optimization techniques. All developed scripts share common parts, which have been extracted into external files. The settings for the script (the configuration file) and the class for creating particles for animation and flow field creation are identical in all tested cases and do not change performance between scripts. Variations in optimized files were made by modifying the draw method (draw method in p5.js defines changes in each animation frame in browser Canvas) through which the animation is updated and displayed in each drawing cycle. Within each draw method of the tested scripts, a flow field is defined. This flow field affects the movement of particles and is influenced by Perlin noise. After setting up the flow field, all particles are iterated through, and the flow field is applied to them, determining their movement, their reaction to the edges of the canvas, and their display on the canvas. The animation visually represents how particles follow the flow field, creating complex patterns as shown on Figure 1.

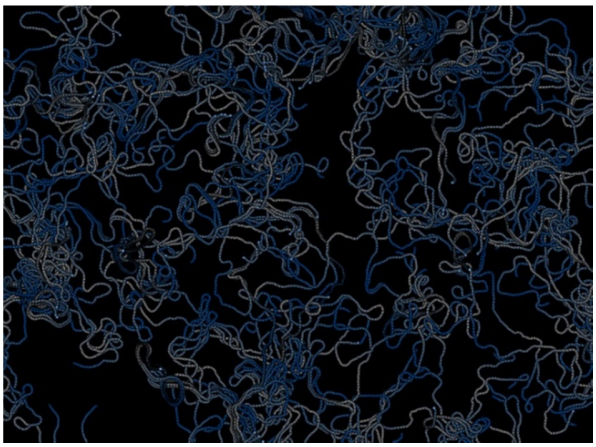

Fig. 1. Created flow field animation used for testing.

Canvas size was set to 800×600 px for testing. The Canvas is divided by 10, both horizontally and vertically to obtain fields for vectors that influence the direction of particle movement. In p5.js, the noise() function is used to generate Perlin noise. The implementation of the noise() function in p5.js supports up to three parameters representing coordinates in the n-dimensional noise space. For testing purposes, xoff and yoff are altered by 0.05 and zoff by 0.01, every time the vector field is generated (in each frame). The speed of the particles during the animation is not differentiated. Particles are defined in the code via a class within which methods that determine the appearance and movement of particles and their display on the Canvas are determined.

During testing, the animation was limited to 15 s, and the number of animated particles varied. The original animation was modified in several ways, but the change was made only within the draw method, in which the way particles are drawn on the screen was altered by applying optimization algorithms. During testing, increasing the number of particles being animated, leads to animation issues, and its framerate decreases. Optimization techniques aim to increase the framerate in animations with many particles. Generally, optimization techniques intend not to draw or delete particles that are not visible on the screen to improve the animation performance. Particles that are hidden from view can be temporarily excluded from the rendering process, thereby reducing the load on the GPU or CPU. For particles displayed as small dots or circles on the screen, a difference which is not visible to the human eye would typically be within a few pixels, especially if the screen is viewed from a normal distance. Threshold in optimising particle visualisation for detecting "overlap" between particles was set to the distance of 1 pixel as the rule for removing or hiding particles. With such intervention, the difference between original and optimized animation is difficult to detect. Over time, more and more particles overlap because they are affected by the same vector field. To prevent a situation in which nearly all particles are removed, a minimum threshold of 1000 particles was established to ensure the animation does not alter to that point. This ascertains that the animation remains as similar as possible to the original. Tests were conducted with the following optimized files:

1. **Array and Array/Delete:** Not displaying or deleting particles if they overlap, checking overlap with arrays. The array of particles is traversed, and only particles that are located more than 1 px away from the particles already placed in the new array are kept. When deleting, the array created ultimately replaces the initial array of particles.
2. **Map and Map/Delete:** Not displaying particles that overlap, with verification via Map. A Map is used to store particles from the original array of particles, with the keys in the Map defined as coordinates rounded to whole numbers. This ensures that particles with identical coordinates are not stored in the Map, thereby reducing the number of particles displayed, especially those that are very close to each other (less than 1 px difference). When deleting particles, an array that replaces the original one is created from the map.
3. **Binning and Binning/Delete** with Map: Binning [14] divides space into rows and columns, directly mapping particles to a grid according to their coordinates. For testing, bins were defined as a Map and the size of bins was set to 50×50 px. Within each bin, another Map is used to store particles with a precision of 1 px. This allows for the removal/deletion of particles that overlap by less than 1 px within the same bin.
4. **Spatial Hashing and Spatial Hashing/Delete:** Spatial hashing [15] deploys the idea of dividing space into fixed "buckets" and then assigning particles to these cells based on their position. This allows for quick searching of neighboring particles without the need to review all particles. Each "bucket" within the spatial hash grid is set to have a size of 50×50 pixels, within which the particles are grouped. For each particle, only other particles within the same bucket need to be checked for particle overlap.
5. **LOD and LOD/Delete:** This approach allows only part of the particles to be displayed in areas of high particle density. In the created animation, each cell for density analysis is 50×50 px and the particle density is analyzed separately for each particle. The density threshold is set to 50 particles per cell. If there are 50 or fewer particles within a cell, all will be displayed, otherwise, a random selection determines which particles will be shown, or in the case of deleting particles, which will be kept.
6. **Quadtree and Quadtree/Delete:** The code initiates a Quadtree [16] with a capacity of 4. Quads are divided into smaller squares only if they are larger than 1 px and if they exceed their capacity. This enables precise organization of particles without unnecessary overlapping. Particles are initially marked as invisible and are then inserted into the Quadtree for more efficient searching. Only particles that are within a defined range (1 px) and marked as visible (only 1 px in range) are exhibited.

The animations were tested on a computer equipped with an Intel® Core™ i7-9750H processor with a 2.60 GHz clock speed, and 16 GB of installed RAM. The system uses a 64-bit version of the Windows 10 Education N operating system. Google Chrome web browser (Ver. 122.0.6261.95, 64-bit) was utilized for executing and testing the animations.

III. TESTING AND RESULTS

During the testing phase, all animations were set to run for 15 s, and during this period, the average framerate of the animation, the total number of frames created, the maximum and minimum framerate, the mode, and the standard deviation were measured. Each animation was run 10 times, and the values presented in this paper are the averages of the conducted measurements. Table I illustrates the results. Results higher than 30 fps are considered as Satisfactory/Optimal framerates, while 30 fps is considered the lower limit for smooth animation in many digital applications. Values from 29-12 fps are somewhat acceptable framerates. They provide less smooth animation but are still capable of conveying motion. In practice, if the animation is not too fast or complex, 12 fps can be acceptable, though some flickering may be noticeable. Values below 12 fps are considered unacceptable.

TABLE I. AVERAGE FRAMERATE FOR ANIMATIONS WITH DIFFERENT NUMBERS OF PARTICLES

	Number of particles in the animation						
	1000	**2500**	**5000**	**7500**	**10000**	**12500**	**15000**
Original	60	37	18	12	9	7	6
Array	60	41	8	3	2	1	1
Array/Delete	60	51	10	4	2	1	1
Map	60	53	28	16	11	9	7
Map/Delete	60	56	41	27	19	13	9
Spatial Hashing	60	51	27	16	11	9	7
Spatial Hashing/Delete	60	54	36	23	13	10	8
LOD	60	44	25	18	15	13	11
LOD/Delete	60	52	41	39	39	39	37
Binning	60	39	20	13	9	8	6
Binning/Delete	60	56	42	25	17	12	9
Quadtree	60	40	20	13	10	8	7
Quadtree/Delete	60	56	44	32	23	15	10

30-60 fps	Satisfactory/Optimal
13-29 fps	Marginal
1-12 fps	Unacceptable

Given the results, it is clear that increasing the number of particles leads to a significant drop in framerate. This expected behavior occurs because a larger number of particles demands more computing resources for processing and displaying. The obtained findings reveal that the animation remains fluid for animations with 1000 particles, regardless of the animation code used. With 2500 particles, all code interventions led to improvements. Beyond 5000 particles, Array and Array/Delete optimization yields unsatisfactory results. In the context of flow field animations, adding an additional layer of logic to manage particle overlap proved to be counterproductive. This becomes particularly pronounced when the particle count is high, as it requires extensive computations to determine which particles not to display or delete. When implementing an array to check for overlaps, each particle is compared to those already added to the new array. As the number of particles increases, the number of comparisons exponentially grows, significantly increasing the total time needed for processing all particles.

Array optimization may be beneficial for a smaller number of particles, where there are fewer comparisons and less memory load. However, with a larger number of particles, the benefits are lost due to raised computational and memory demands, suggesting that such a form of optimization should be avoided in these cases. Other techniques proved useful in maintaining a higher framerate compared to the original approach with a larger number of particles.

Methods that involve deleting overlapping particles tend to maintain a higher framerate due to the elimination of the overlapping particles which reduces the number of particles to 1000, essentially restoring fluid animation. From this, it can be concluded that in situations with a high density of particles, managing memory and the number of active objects is key to performance. The greatest impact on performance will come from reducing the number of particles that need to be animated.

Optimization using Maps for filtering and displaying particles demonstrated significantly better results compared to optimization using Arrays. This improved performance can be attributed to the more efficient overlap check using Maps. Defining Map keys as spatial coordinates ensures that particles with identical coordinates are not stored multiple times. This automatically filters particles that are very close to each other, reducing the total number of particles to be displayed and facilitating the deletion of hidden particles. Using Maps eliminates the need for double loops and decreases the computational complexity of the process. The efficiency of Maps comes from their ability to provide quick access and data updates, significantly speeding up the decision-making process on which particles to retain for display. Although this method does not utilize complex optimization algorithms, its simplicity and efficiency in practice make it a good option. Deploying Maps to filter particles by coordinates can achieve significant performance improvements with relatively simple logic.

Table II displays the results of performance calculations after code optimization. The table depicts the percentages of improvement calculated using the formula $((FPS_{optimised}-FPS_{original})/ FPS_{original})\times100\%$. Improvement is classified as minimal, significant, or optimal depending on its percentage. Any improvement that is statistically significant and consistently reproducible can be considered beneficial. Improvements of 20% or more are regarded significant and can greatly enhance user experience, reduce the load on hardware resources, or enable additional functionalities that were not feasible before due to performance limitations. Improvements of 50% or more are thought as optimal and often result in significant changes in application performance. Such improvements can change the way an application is used, enabling more complex simulations, faster data processing, or smooth animations in high resolution. For highly interactive applications or games, a framerate improvement leading to smoother display can have a direct impact on user experience and is thus considered highly valuable. Ultimately, the goal is to achieve a balance between the costs of optimization (including the time and resources needed for implementation) and the benefits that this optimization brings in terms of performance and user experience.

TABLE II. PERFORMANCE OPTIMIZATION PERCENTAGES FOR ANIMATIONS WITH VARYING NUMBERS OF PARTICLES

	Number of particles in the animation					
	2500	**5000**	**7500**	**10000**	**12500**	**15000**
Array	11	-56	-75	-78	-86	-83
Array/Delete	38	-44	-67	-78	-86	-83
Binning	5	11	8	0	14	0
Quadtree	8	11	8	11	14	17
Spatial Hashing	38	50	33	22	29	17
Map	43	56	33	22	29	17
LOD	19	39	50	67	86	83
Spatial Hashing/Delete	46	100	92	44	43	33
Binning/Delete	51	133	108	89	71	50
Map/Delete	51	128	125	111	86	50
Quadtree/Delete	51	144	167	156	114	67
LOD/Delete	41	128	225	333	457	517

> 50%	Optimal Improvement
20-50%	Significant Improvement
1-20%	Minimal Improvement
< 0%	No change or deterioration in performance

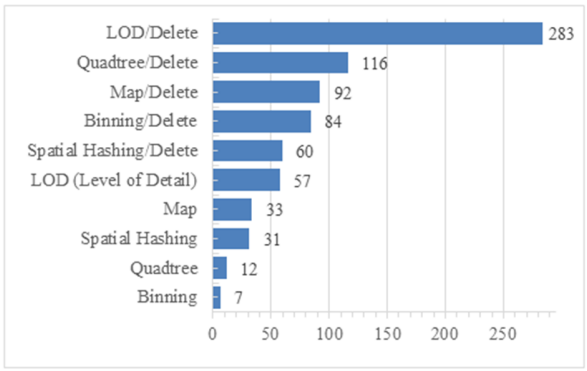

Fig. 2. Average optimization percentage.

Figure 2 portrays a comparison of the average optimization for all measurements. It is evident that all Delete algorithms bring optimal improvements. Generally, it can be concluded that methods involving particle deletion show better performance as the number of particles increases. This is because delete techniques reduce the number of particles for the animation, thereby decreasing the complexity of the code being executed. The LOD/Delete method exhibits exceptionally good performance stability and the best results from those of all the tests. With LOD/Delete, even at the highest number of particles, the framerate remains high. LOD/Delete has proven to be the most effective technique, with high percentages of improvement even at the highest number of particles. The LOD approach has proven very effective in dynamically adjusting animation details according to the available resources. This technique provides the best improvements due to the modified deletion technique. With this algorithm, deletion is determined by the density of particles in a specific area, and is randomly performed. The particular technique, which differs from other algorithms that precisely delete particles, requires fewer comparisons to exactly determine which particles need to be deleted, thus optimizing

the code to the greatest extent. Such a code modification also changes the way the animation is displayed, as random deletion of particles is noticeable in the animation. The best optimization of the animation is achieved with LOD/Delete but it sacrifices precision in deleting particles and losses in the visual quality of the animation occur. Adaptive techniques like LOD, which adjust the level of detail in real-time, can provide consistent performance even under heavy load.

If there is a need for deletion precision to be maintained, Quadtree/Delete has proven to be very effective, especially with a higher number of particles, as it shows a significant improvement in performance compared to the original animation. Interesting results are also provided by Map/Delete, where high code execution optimization is achieved without the implementation of an optimization algorithm but only by changing the code execution logic. This technique demonstrates consistent improvements with an increasing number of particles, indicating the efficiency of using the Map to eliminate overlapping particles.

Among the techniques that do not delete particles and only hide them from the animation, LOD, Map, and Spatial Hashing have proven to be the best. If the precision of the animation is not important, LOD provides optimal optimization while Map and Spatial Hashing offer satisfactory results. From the obtained results shown in Figure 3, it is evident that techniques, such as LOD/Delete, Quadtree/Delete and Map/Delete reveal higher standard deviations during execution. In the case of Delete scripts, the higher standard deviation and oscillations in script execution can be attributed to changes in the framerate that occur due to the deletion of particles, i.e. as the number of particles in the visualization decreases, the framerate increases. This is precisely why these visualizations give a large standard deviation. Standard variation and framerate are anticipated to normalize through script execution, aiming for an average framerate of 60 fps to ensure smooth and fluid animations.

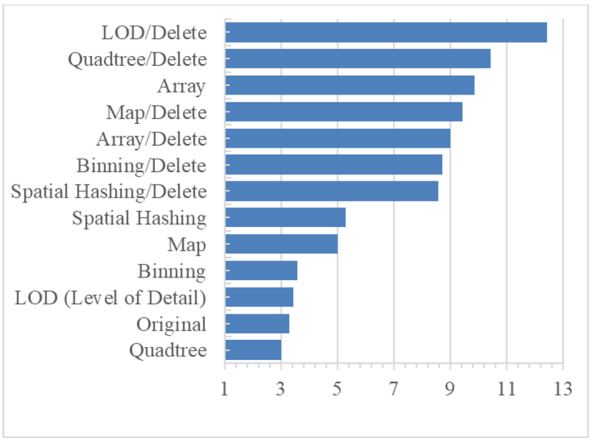

Fig. 3. Average standard deviation in fps across all tests for each optimization script.

In standard animations, variations in execution and variable framerate can result from several factors, namely source code complexity, memory and processor efficiency, and other system resources. A high standard deviation may suggest that animations vary in predictability, potentially affecting user experience. Due to variations in performance, some frameworks may run smoothly, while others may experience stutters or delays, depending on system load and code efficiency. Higher standard deviation indicates unpredictable performances, which can be problematic from the point of view of user experience or interactive visualizations. An example of large fps variations is shown by Array, where problems arise due to inefficient use of resources.

Key conclusions were drawn from the conducted research and comparisons. These can be applied to other code visualizations and optimizations, regardless of the specific implementation details or technology used:

- Optimized computation of complex visualizations is important to ensure smooth interaction and user experience. Various optimization strategies are key to achieving this goal.
- Optimizations that enable dynamic adaptation under increased load can enable high performance.
- Reducing the load on processing and memory resources ensures fluid visualizations. With demanding calculations and complex scripts, it is necessary to optimize the management of data sets and avoid unnecessary processing.
- Application of various optimization techniques and algorithms can significantly improve visualization performance. When choosing optimization techniques, it is necessary to strategically select and combine different approaches, with continuous evaluation and adaptation based on specific project requirements and constraints.
- By adjusting the LOD in rendering, based on density and random deletion, visualizations can be greatly optimized with less change in visual quality for the user.
- The implementation of Quadtree algorithms for the organization of particles and more efficient detection of overlaps or collisions permits the creation of more fluid visualizations.
- Utilizing the Map structure in JavaScript promotes fast filtering and comparison of data, while reducing the need for double loops and computational complexity, allowing for fast data access and updates.

The findings of this study support the hypothesis that optimization techniques can significantly improve (more than 500% in the optimal cases) rendering performance in browser environments. The obtained results emphasize the wider applicability and potential of optimization methods in improving visualizations based on web technologies.

This paper contributes to the academic discourse on code optimization and provides practical insights and guidance for developers and researchers investigating complex visualization tasks and their performance. Although some techniques have demonstrated remarkable results, it is important to continue research and experimentation. It is necessary to discover new models and approaches to optimization, which can be achieved

by combining different techniques or developing new algorithms.

The application, comparison, and evaluation of various optimization techniques within the p5.js [17-19] environment was performed in an innovative way, including Quadtree, Spatial Hashing, Binning, LOD adjustments, and the use of the Map data structure. This detailed analysis presents a new approach in optimizing the performance of complex animations in web browsers, demonstrating how advanced optimization techniques can be successfully applied and adapted to improve performance in p5.js applications. This paper presents specific recommendations for the implementation of these optimization techniques, thus providing practical value for developers and researchers engaged in the creation of complex visualizations. By comparing the performance of different optimization approaches, the most efficient techniques were identified, thereby providing important guidelines that can contribute to more effective coding and a better user experience. The discovery that techniques like LOD/Delete provide significant performance improvements even under high load is another important contribution of this paper. This finding highlights the potential of adaptive optimization techniques in dynamic web environments, which may inspire further research and development in the field of rendering optimization. Finally, the application of specific optimization algorithms within the browser environment, especially applying p5.js, adds new dimensions to the existing literature. Not only does this research demonstrate how advanced optimization techniques can be successfully implemented in p5.js applications, but also encourages further exploration and innovation in pedagogical approaches to computer graphics programming and learning, with a particular focus on performance and optimization. With these contributions, this work is positioned as a significant reference in continuous efforts to improve the efficiency and performance of web-based visualizations, contributing to both the academic and practical spheres of software development.

IV. CONCLUSION

The acquired results show significant improvements in rendering performance through the application and adaptation of various optimization techniques in the p5.js environment. The findings not only confirm the effectiveness of our approaches, but also contribute to the existing literature by offering new insights into the application of optimization strategies in the specific contexts of web development, digital art and education.

In order to execute complex computer calculations and perform visualizations and animations, it is recommended to use optimization techniques and algorithms as displayed in this paper. As a result of the optimization, the original tested animation was improved by over 500% in its performance. The percentage of improvement varied depending on the applied optimization algorithm. Techniques that can be applied when optimizing the code include the application of various optimization algorithms, the development of algorithms related to the problem being solved, or the application of code improvement using optimized data structures such as Maps.

This research contributes to the academic discussion on optimization. From the acquired results, practical tips and guidelines can be drawn for developers and researchers involved in solving complex optimization tasks. The novelty of the conducted research lies in the application of optimization techniques within the browser environment using the p5.js library, JavaScript code, and HTML Canvas. The p5.js library has not been thoroughly explored for such purposes. Further research could be aimed at researching and achieving even greater efficiency when applying optimization techniques, which could be accomplished by applying dynamic and adaptive algorithms in web browsers.

The conducted research and the obtained results indicate the fact that the application of optimization techniques can crucially improve performance efficiency, reduce the load on resources, and ameliorate user experience. The integration of optimization techniques into the program code can affect the possibility of developing more complex and resource-efficient web applications. The application of optimization algorithms in p5.js also opens space for innovation in pedagogical approaches to learning programming and computer graphics, where performance and optimization topics could be studied.

REFERENCES

[1] R. V. V. Krishna and S. Srinivas Kumar, "Hybridizing Differential Evolution with a Genetic Algorithm for Color Image Segmentation," *Engineering, Technology & Applied Science Research*, vol. 6, no. 5, pp. 1182-1186, Oct. 2016, https://doi.org/10.48084/etasr.799.

[2] B. K. Alsaidi, B. J. Al-Khafaji, and S. A. A. Wahab, "Content Based Image Clustering Technique Using Statistical Features and Genetic Algorithm," *Engineering, Technology & Applied Science Research*, vol. 9, no. 2, pp. 3892-3895, Apr. 2019, https://doi.org/10.48084/etasr.2497.

[3] M. Platings and A. M. Day, "Compression of large-scale terrain data for real-time visualization using a tiled quad tree," *Computer Graphics Forum*, vol. 23, no. 4, pp. 741-759, Dec. 2004, https://doi.org/10.1111/j.1467-8659.2004.00806.x.

[4] W. Duan, J. Luo, G. Ni, B. Tang, Q. Hu, and Y. Gao, "Exclusive grouped spatial hashing," *Computers & Graphics*, vol. 70, pp. 71–79, Feb. 2018, https://doi.org/10.1016/j.cag.2017.08.012.

[5] M. Trapp and J. Döllner, "Interactive Close-Up Rendering for Detail+Overview Visualization of 3D Digital Terrain Models," in *2019 23rd International Conference Information Visualisation (IV)*, Paris, France, Jul. 2019, pp. 275–280, https://doi.org/10.1109/IV.2019.00053.

[6] S. Weiss and R. Westermann, "Differentiable Direct Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 562-572, Jan. 2022, https://doi.org/10.1109/TVCG.2021.3114769.

[7] Q. Zaheer and J. Masud, "Visualization of flow field of a liquid ejector pump using embedded LES methodology," *Journal of Visualization*, vol. 20, no. 4, pp. 777-788, Nov. 2017, https://doi.org/10.1007/s12650-017-0429-3.

[8] M. Auer and A. Zipf, "3D WebGIS: From Visualization to Analysis. An Efficient Browser-Based 3D Line-of-Sight Analysis," *ISPRS International Journal of Geo-Information*, vol. 7, no. 7, Jul. 2018, Art. no. 279, https://doi.org/10.3390/ijgi7070279.

[9] M. Emelianenko, "Fast Multilevel CVT-Based Adaptive Data Visualization Algorithm," *Numerical Mathematics-Theory Methods and Applications*, vol. 3, no. 2, pp. 195-211, May 2010, https://doi.org/10.4208/nmtma.2010.32s.5.

[10] B. Subbaraman, S. Shim, and N. Peek, "Forking a Sketch: How the OpenProcessing Community Uses Remixing to Collect, Annotate, Tune, and Extend Creative Code," in *Proceedings of the 2023 ACM Designing Interactive Systems Conference*, New York, NY, USA, Apr. 2023, pp. 326–342, https://doi.org/10.1145/3563657.3595969.

[11] C. Orban, C. D. Porter, N. K. Brecht, R. M. Teeling-Smith, and K. A. Harper, "A novel approach for using programming exercises in electromagnetism coursework," in *2017 Physics Education Research Conference Proceedings*, Mar. 2018, pp. 288–291, https://doi.org/10.1119/perc.2017.pr.067.

[12] C. Gasch, M. Chover, I. Remolar, and C. Rebollo, "Procedural modelling of terrains with constraints," *Multimedia Tools and Applications*, vol. 79, no. 41-42, pp. 31125-31146, Aug. 2020, https://doi.org/10.1007/s11042-020-09476-3.

[13] W. Lefer, B. Jobard, and C. Leduc, "High-quality animation of 2D steady vector fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 1, pp. 2-14, Jan.-Feb. 2004, https://doi.org/10.1109/TVCG.2004.1260754.

[14] "Binning in Data Mining," *GeeksforGeeks*. https://www.geeksforgeeks.org/binning-in-data-mining/.

[15] "Spatial Hashing," *GameDev.net*. https://gamedev.net/tutorials/programming/general-and-gameplay-programming/spatial-hashing-r2697.

[16] "Quad Tree," *GeeksforGeeks*. https://www.geeksforgeeks.org/quad-tree/.

[17] "Optimizing p5.js Code for Performance," *GitHub*. https://github.com/processing/p5.js/wiki/Optimizing-p5.js-Code-for-Performance.

[18] "Overview. A short introduction to the Processing software and projects from the community," *Processing*. https://processing.org//overview/.

[19] "Home," *p5.js*. https://p5js.org/.