http://doi.org/10.35784/iapgos.3828

received: 03.07.2023 | revised: 06.09.2023 | accepted: 27.09.2023 | available online: 30.09.2023

RESEARCH ON CALCULATION OPTIMIZATION METHODS USED IN COMPUTER GAMES DEVELOPMENT

Nataliia Fedotova[1], Maksim Protsenko[1], Iryna Baranova[1], Svitlana Vashchenko[1], Yaroslava Dehtiarenko[2]

[1]Sumy State University, Faculty of Electronics and Information Technologies, Department of Information Technology, Sumy, Ukraine, [2]Lublin University of Technology, Faculty of Electrical Engineering and Computer Science, Lublin, Poland

Abstract. *In the field of computer game development, there are numerous optimization methods that help to significantly reduce the number of calculations that the game system performs while playing the game. In its turn, this allows to display increasingly realistic graphics. The paper presents the performed analysis of general optimization methods used in the game engine Unreal Engine, such as Distance Culling, Occlusion Culling, Frustum Culling, LODs, Level Streaming, and the Nanite System. The main factors such as resource intensity, visual quality, number of objects, and scale have been determined. The research results demonstrate that properly applied optimization methods can improve game performance and reduce the computational load on the system, which is crucial both functionally and aesthetically.*

Keywords: UE5, optimization methods, games development

BADANIE METOD OPTYMALIZACJI OBLICZEŃ STOSOWANYCH W TWORZENIU GIER KOMPUTEROWYCH

Streszczenie. *W dziedzinie tworzenia gier komputerowych istnieje wiele metod optymalizacyjnych, które pozwalają znacznie zredukować liczbę obliczeń, jakie wykonuje system gry podczas renderowania, co z kolei pozwala na wyświetlanie coraz bardziej realistycznej grafiki. W pracy przeprowadzono analizę ogólnych metod optymalizacji stosowanych w silniku gier Unreal Engine, takich jak Distance Culling, Occlusion Culling, Frustum Culling, LODs, Level Streaming oraz Nanite System. Zidentyfikowano główne czynniki, które mają znaczenie: zużycie zasobów, jakość wizualna, liczba obiektów i skalowalność. Wyniki badań pokazują, że prawidłowo zastosowane metody optymalizacji mogą poprawić wydajność gry i zmniejszyć obciążenie systemu obliczeniowego, co jest istotne zarówno pod względem funkcjonalnym, jak i estetycznym.*

Słowa kluczowe: UE5, metody optymalizacji, tworzenie gier

Introduction

A computer game is a complex program that involves a large number of mathematical calculations. Each object in the game has its own set of unique geometric (shape, number of polygons) and physical parameters (weight, size, density) and optical properties of the object's surface, which characterize the its ability to reflect light and shadow. Additionally, sometimes it is necessary to consider the partial or complete destruction of the object. Each of these aspects requires to be calculated at least once for each frame, which are sequentially displayed on the monitor. Depending on the game's characteristics, this can occur at 30, 60, 120, or even 240 frames per second (fps).

In modern games, the number of objects with the same model can reach tens of thousands, and the number of polygons on a single model can reach millions. The resources of gaming devices are not infinite, which necessitates reducing the load on the computer's hardware resources used by the game. At different stages of game development, optimization methods corresponding to that stage are employed. The developer must understand how and which optimization method or their combination affects the game's performance. Performance optimization in games aims to reduce the time required to display a game frame and increase the number of frames per second. This ensures a more convenient and smooth gaming experience for the user and overall comfort during gameplay.

This research aims to examine the performance optimization methods used in the computer game industry and to analyze the most popular and effective of them.

A game prototype using the Unreal Engine 5 (UE5) game engine environment, where optimization methods will be tested, was developed to evaluate the effectiveness of existing methods. The testing will be conducted in two scenarios. The first one will be carried out without utilizing optimization methods. And the second scenario will be performed with implementing the optimization method. Each method will be tested separately.

When analyzing the methods' effectiveness, it is essential to consider the following aspect. Practically all existing or new games contain many geometric objects and their textures, complex shaders for model visualization, and numerous intricate visual effects. However, computers have certain technical limitations, such as available memory capacity for the game, frame rate restrictions, or limits on the number of objects that can be displayed on the screen. Therefore, solving this issue is urgent.

1. Optimization methods

It should be noted that all users' computers significantly differ in their efficiency. However, they are all united by the fact that the fewer calculations will be performed by the device, the better. One of the key tasks during the game development is to create a smooth game process. The game should be uninterrupted, and any delays should only be caused by the plot or game mechanics. Optimization methods are used exactly to achieve such gameplay by minimizing delays in calculations.

Optimization methods depend on the game engine used for the game development. For example, let's consider the most popular game engines as Unity and Unreal Engine. While Unity is more oriented towards simple indie games and mobile applications, Unreal Engine aims to create more labor-intensive games with many calculations and realistic graphics. Further, we are going to discuss optimization methods related exactly to Unreal Engine.

According to research materials [1, 5], the following main optimization methods are defined in Unreal Engine:

- Distance culling;
- Occlusion culling;
- Frustum culling;
- LODs;
- Level streaming;
- Nanite System.

Let's consider each of them in detail.

Distance culling. The essence of this method is that objects stop being displayed when the camera is at a certain distance from them. The distance can be set for each object or a group of objects.

Occlusion culling. In this method objects, which are not within the camera's field of view, are culled. A preload time is set in advance to start loading the objects slightly earlier, ensuring they are loaded by the time the player (camera) looks at them [10]. Figure 1 illustrates an example of disabling objects which are outside the player's field of view for a better understanding of the method.

Frustum culling. The principle of this method is very similar to Occlusion culling' one, but objects, which overlap by other objects, are not drawn on the screen.

In other words, the player does not see particular objects covered by other objects (Fig. 2).

LODs (Level of Detail). This method reduces the number of geometric elements that must be processed for displaying on the screen. Instead of fully loading all the fine object details, its less detailed versions are loaded depending on the distance to the object.

The model becomes more detailed when the player is close to the object and can see it clearly [6, 13] (Fig. 3).

Level streaming. This method is used to reduce memory load. It allows loading the game parts, which are the closest to the player, while other elements, which are far away, are not loaded.

If everything is done correctly, this allows creating the extensive seamless maps where the players can feel themselves like playing in a world which impresses with its scale [14]. The Level Streaming method is represented schematically by Fig. 4.

The Nanite System. This method is based on geometric objects' discretization, which allows rendering highly detailed scenes with billions of polygons in real-time.

During camera zooming out from the object, the Nanite System merges multiple model polygons into one, thereby significantly reduce the system load.

Benefits of Nanite: There is no need to optimize models according to fewer polygons. The system handles it automatically. It became possible directly to import high-quality output models, such as ZBrush sculpts and scanning the photogrammetry. Level of Detail (LOD) is processed automatically and no longer requires manual setting for individual LOD mesh. Quality loss is rare or nonexistent, especially with LOD transitions [15]. Despite the benefits, there are practical limitations to the Nanite System which remain in recent versions. For example, the objects amount with a single model in the scene, triangles amount per the mesh, material complexity, output resolution as well as efficiency should be carefully measured for any combination of content and hardware.

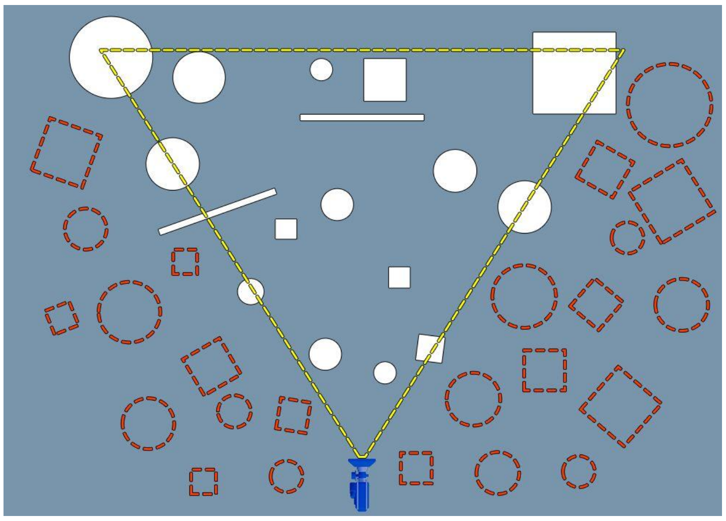

Fig. 1. Occlusion culling principle of operation [12]

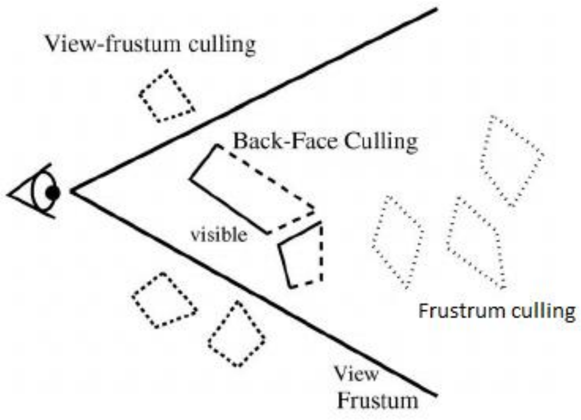

Fig. 2. Frustum culling principle of operation [8]

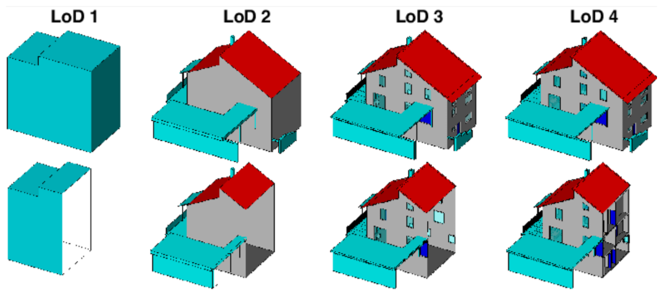

Fig. 3. Using LOD levels [2]

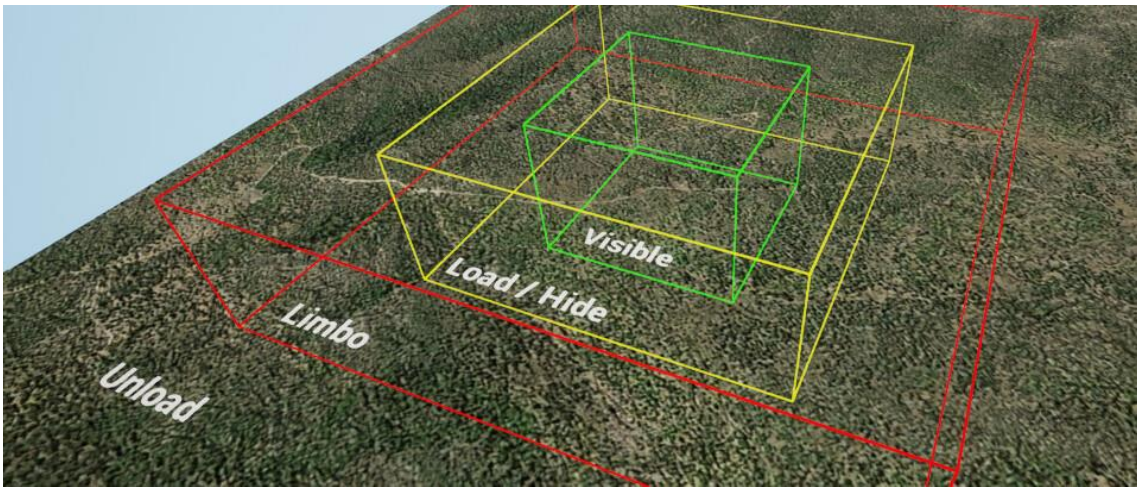

Fig. 4. Level streaming principle of operation [14]

2. Related work

The results of application the optimization methods may differ and strongly depend on the project itself. One team can bet on optimization immediately during the project development. Otherwise, the game may resemble a "dump" that will have to be sorted out for a long time. In this case, while using optimization methods, the system load can be reduced tenfold.

There is still no universal answer regarding which of the existing methods should be used for a specific project. It all depends on the game and the result the developers want to achieve. If the game is exceedingly small, it may not be optimized. If the volume is average, the simplest optimization methods can be used, such as Distance culling, Occlusion culling and Frustrum culling [7]. For large projects with various objects and large maps, it is advisable to utilize a combination of all optimization methods immediately. When dealing with a large number of identical objects in the scene, it is worth thinking about using the Nanite System [9]. In some cases, only this system can increase the number of fps (frames per second) by multiple times.

If we pay attention to a specific situation, in his book Chris Dickinson describes a case where he and his team had to optimize a particular game. In that case, it was possible to increase the average frame rate in the game from 30 to 120 fps (tests were conducted on the same computer with the same settings) [4]. What's interesting is that he didn't use all the optimization methods. He just applied the LODs and set them correctly. Since the project had many high-poly models, the increase in the frames number was fantastic. They also used other optimization methods in the game, but Chris's decision to integrate LODs played a significant role in achieving such a remarkable result.

Intel's research with Unreal Engine 4 shows a 12.2% increase in fps. Optimization was carried out only by optimizing the models and textures [16]. The same optimization principle was applied in Alberto Alvarez's work, where a simple 2D game achieved a 5% improvement in fps [3].

Certain specific games often use the same specific optimization techniques, as described in Hai Xu's article on optimizing a 3D city in Unreal Engine 5. He describes an interesting system where the city models are replaced with a volumetric map for optimization. This map consists of a city image overlaid on a height map, creating a visually appealing representation of the city (as long as the camera is not zoomed in too closely). This approach significantly increased the frame rate for his project (over 224%) [11].

Therefore, this research has been conducted to analyze and compare the effectiveness of existing optimization methods.

3. Research of optimization methods using a UE5 game prototype

3.1. Initial testing conditions

To systematize the results, we used the MSI Afterburner program [17], which provides statistics of the load on the processor, RAM, graphics adapter (video card), and the number of frames per second (fps). Additionally, the Unreal Engine's built-in tools were used to display CPU and GPU load statistics. The computer characteristics, using which the optimization methods were tested, are given in table 1.

To assess the effectiveness of the Occlusion culling, Distance culling, Frustrum culling, and Level Streaming methods, a game level was developed. It consists of a set of simple geometric shapes that have minimal impact on system load and ten fire sources, which heavily stress the graphics card. The result of applying these methods is demonstrated by exactly these fire sources. Testing the Nanite System and LODs methods was conducted on a game level which contains 1000 identical high-detailed objects. The objects of the Nanite System have 38,099 polygons, while the LODs have 960 polygons.

Table 1. Hardware characteristics

CPU	Ryzen r7 5800x
GPU	GTX 1660 super
RAM	32 Gb DDR4 3600 MHz
SSD	MSI M390 – 1Tb
Monitor	Full HD (1920x1080)

3.2. Researching the impact of the Occlusion culling method

Occlusion culling is already implemented and enabled by default in UE5. We can only turn it off and compare the characteristics of the system load with the method on and method off. To disable it, the "Occlusion Culling" checkbox should be deactivated in the project settings in the "Rendering" tab.

3.3. Researching the impact of the Flustrum culling method

Flustrum culling is also implemented and enabled by default in UE5. During the testing of Frustum culling, similar results were obtained compared to the previous tests because the operation principle is identical. The testing was conducted on the same level as previously and involved disabling objects which were not visible to the player.

Based on the increasing in frames per second, we can conclude that the load on the graphics card at least halved by using Frustum culling. Without optimization, in average we had forty frames per second, and with optimization, it increased to 90 frames per second. Since the graphics card was fully utilized by 100%, it was the main factor limited the system from generating more frames per second.

3.4. Researching the impact of the Distance Culling method

Initially, we have checked how the system behaves if the player moves far away from the load source. The system load is slightly lower compared to being close to the fire source.

Distance culling in UE5 can also be implemented, but it needs to be enabled manually. It can be configured specifically for the fire by setting each actor's Desired Max Draw Distance setting.

When we are at the specified distance from the fire, the effect disappears, and the system load returns to the level observed in the previous methods.

3.5. Researching the impact of the LODs method

To use this method, you have to create a set of LODs (Level of Detail) for the model that needs setting. This time, the test level consists of a substantial collection of spheres (2120 units), the existence of which loads the system by the calculations of light, physics, etc. Five levels of LODs were created for the sphere model using the built-in tools of UE5. The first level of the sphere model has 960 polygons, while the last one has 120 polygons.

The engine automatically switches LODs regarding the player's distance from a particular object. It can also be configured within the LOD creation menu using default parameters. For this research the system load has been tested using the default settings.

Comparison the testing results of all the mentioned optimization methods are presented in table 2. As we can see, using the LODs method results in a relatively slight decrease in system load. However, we obtained a significantly higher frame rate. This is because the RAM bandwidth of the device limited us due to using 2120 models. And since using LODs in real-time playback mode, the model's size decreases, and the load on the RAM bandwidth also decreases. This allows the system to utilize its potential better, increasing system load while achieving a higher frame rate.

3.6. Researching the impact of the Level streaming method

To investigate the impact of the Level Streaming method, we have created two sub-levels in Unreal Engine 5. The first sub-level contains elements with a high load on the graphics card, such as fire simulation or complex graphics. The second one has a trigger that disables the previous sub-level after the player reaches a certain point in the game.

This has allowed us to measure the sub-level loading time and the impact of the Level Streaming method on game efficiency. The general view of the scene is shown by figure 5. Sub-level 2 contains fire effects which cause significant resource load on the computer.

The switching of sub-levels is implemented using a trigger. The logic for the switching was described in the Blueprint of the level that contains two sub-levels (Fig. 6).

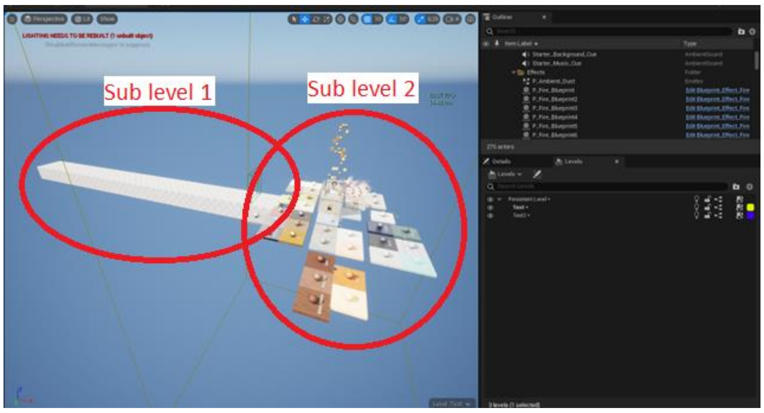

Fig. 5. General view of the level

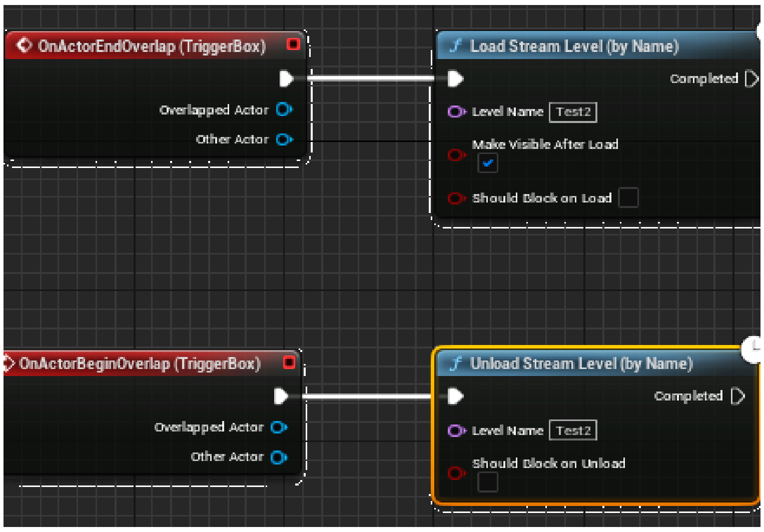

Fig. 6. Level Streaming logic implementation

Table 2. Test results

	Occlusion Culling		Flustrum culling		Distance culling		LODs		Level streaming		Nanite system	
Method usage status	Off	On	Off	On	Off	On	Off	On	Off	On	Off	On
CPU load (%)	2%	3%	2%	3%	2%	3%	5%	7%	3%	1%	3%	2%
Video card load (%)	100%	99%	100%	99%	100%	99%	39%	49%	100%	98%	100%	98%
RAM usage (MB)	17690	17773	17690	17334	17161	16792	15642	17919	17566	17556	19472	20148
The coefficient of load on the graphics card	25,27	12,78	25,27	10,62	15,98	11,22	6,7	11,89	18,7	10,27	14,92	10,66
Average number of fps	39	78	39	93	62	93	36	46	53	92	66	92
Download time (seconds)	0.3	0.3	0.3	0.3	0.3	0.23	2.65	3.05	0.3	0.3	0.33	0.45

The logic implementation involves disabling one sub-level under certain conditions, in our case, when we go beyond its boundaries. The high-load effects are located in the second sub-level, so the load decreases when we switch to the first sub-level.

When any item hits the trigger, sub-level 2 disappears, reducing the system load. Disabling the sub-level with high graphics load (fire effect) has significant impact and reduces the system load.

3.7. Researching the impact of the Nanite system method

To test the Nanite system optimization method, we have used a stone model which has 800 triangles (polygons). The Nanite system is already implemented in the engine.

We have enabled it for the specific model, waited for shaders compilation, and tested the changes. The obtained result, according to terms of frames per second, does not differ from the initial one. However, both load on the graphics card and the frame creation time have significantly decreased.

Therefore, we can conclude that the load on the graphics card has been reduced by half, and the frame rate is limited by the PC's memory or the engine itself (due to the enormous number of objects in the scene).

We have used a high-quality model with 38,099 polygons to simulate a higher load on the graphics card. We have added 100 instances of this model to the scene and observed the results without optimization (Table 2). As we can see both an increase in RAM usage and an increase in the number of frames per second take place. Thus, the Nanite System is a new way to optimize models. Although, it is not sufficiently developed for games.

It works perfectly when using high-poly models with large file sizes on the hard disk. While this may be fine in film production, it poses game challenges. In games, it is still simpler and more practical to use LOD systems, which can be generated easily in just a few clicks in UE5.

4. Discussion

Based on the information described above, game optimization is a complex process which even may not be necessary applied to all games. Game projects with various gameplay designs require developers to create combinations of optimization methods to ensure optimal game operation on different devices and for diverse audiences.

As the tendency shows, optimization is an important aspect of modern game development. It is essential to develop techniques for graphics optimization which allow reducing the load on calculations resources and maintaining game efficiency.

In addition, the increasing scale of game projects and demand changing for several games types lead to need of using the optimization to increase efficiency and provide maximum comfort for players. To achieve these aims, developers use a wide range of optimization techniques, including optimizing the processes of physics processing, reducing the number of interactions with objects in the scene, and utilizing interim software.

5. Conclusions

According to the review of optimization methods applying in the game development, it can be stated that the optimization process requires a comprehensive approach for analyzing the project and finding the most problematic areas. Reviewing popular and effective optimization methods is useful, as game development is. Based on the results of this research, it is possible to conclude the following desired scope of using the optimization methods, which were introduced within the paper framework:

- Occlusion Culling and Frustum Culling should be used in all games which render sprites and models to prevent excessive device load due to rendering these graphics.
- Distance Culling is helpful in games with open-world environments where the player can be far away from particular objects and can be turned off.
- LODs are useful for games which have detailed models.
- Level Streaming is good for large levels which may be problematic to display completely.
- Nanite System is appropriate for large projects with many high-poly models which do not have time for optimization (especially for cinematography).

References

[1] Akenine-Moller T. et al.: Real-Time Rendering. 4th ed., A K Peters/CRC Press, 2018. 1200.
[2] Akmalia R. et al.: TLS for generating multi-LOD of 3D building model. IOP Conference Series: Earth and Environmental Science 2014.
[3] Alvarez A.: Exploring Game Design through Human-AI Collaboration, 2022.
[4] Dickinson C.: Unity 5 Game Optimization. Packt Publishing, 2015.
[5] Gregory J.: Game Engine Architecture. 3rd ed., CRC Press, 2019.
[6] Hasenfratz J.-M. et al.: A survey of Real-Time Soft Shadows Algorithms. Computer Graphics Forum 4(22), 2003, 753–774.
[7] Hogan J. et al.: Analyzing Performance Issues of Virtual Reality Applications. ArXiv 2022, (abs/2211.02013).
[8] Johansson M., Roupé M., Bosch-Sijtsema P.: Real-time visualization of building information models (BIM). Automation in Construction 54, 2015.
[9] Penty C.: Behind the Scenes of The Cavern UE5 Cinematic Visual Tech Test SIGGRAPH ’22. New York, USA: Association for Computing Machinery, 2022.
[10] Sekulic D.: Efficient Occlusion Culling Addison-Wesley Professional, 2018.
[11] Xu H. et al.: Efficient visualization of 3D city scenes by integrating the GIS and Unreal Engine. SPIE, 2023. 125510I.
[12] Unreal Engine 5 Documentation: Visibility and Occlusion Culling. https://docs.unrealengine.com/5.1/en-US/visibility-and-occlusion-culling-in-unreal-engine/ (available: 02 12.07.2022).
[13] Unreal Engine 4 Documentation: Creating and Using LODs. https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Types/StaticMeshes/HowTo/LODs/ (available: 12.07.2022).
[14] Unreal Engine 4 Documentation: Level Streaming Overview. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LevelStreaming/Overview/ (available: 20.07.2022).
[15] Unreal Engine 5 Documentation. Nanite Virtualized Geometry. https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/ (available: 20.07.2022).
[16] Unreal Engine 4 Optimization Tutorial, Part 1: https://www.intel.com/content/www/us/en/developer/articles/training/unreal-engine-4-optimization-tutorial-part-1.html (available: 20.09.2022).
[17] MSI Afterburner: https://ua.msi.com/Landing/afterburner/graphics-cards (available: 20.09.2022).

Ph.D. Nataliia Fedotova
e-mail: n.fedotova@cs.sumdu.edu.ua

Associate professor of the Department of Information Technology, Sumy State University, Ukraine. Author and co-author of more than 30 scientific papers. The author`s research area focuses on e-learning systems, information technologies of design and management in complex systems, 3D modeling, visualization and animation, real-time computer graphics and simulations.

http://orcid.org/0000-0001-9304-1693

M.Sc. Maksim Protsenko
e-mail: protsenko85g@gmail.com

Bachelor and master of the Department of Information Technologies, Faculty of Electronics and Information Technologies, Sumy State University. Middle C++ developer at MoonMana Games.

http://orcid.org/00009-0008-9575-8549

Ph.D. Iryna Baranova
e-mail: i.baranova@cs.sumdu.edu.ua

Associate professor of the Department of Information Technology, Sumy State University, Ukraine. Author and co-author of more than 30 scientific papers. Research interests: 3D modeling, visualization, animation, software application for the development of systems of automated design, e-learning systems.

http://orcid.org/0000-0002-3767-8099

Ph.D. Svitlana Vashchenko
e-mail: s.vashchenko@cs.sumdu.edu.ua

Associate professor of the Department of Information Technology, Sumy State University, Ukraine.
Research interests: software engineering, use of modern information technologies (in particular, system modeling, structural-functional analysis) for the development of application software in various fields of activity, e-learning systems, and visualization.

http://orcid.org/0000-0002-7021-2629

Yaroslava Dehtiarenko
e-mail: s99277@pollub.edu.pl

Student of the Faculty of Electrical Engineering and Computer Science, Lublin University of Technology, Poland. The author`s research area focuses on optimization methods, human-computer interaction, real-time systems, computer graphics and simulations.

http://orcid.org/0009-0004-9455-7092