Testing of Vulkan Visualization for Geo-Models on Mobile Devices and Desktop Systems with Ray Tracing GPUs

M. Mustafin[2], O. Turar[1*], D. Akhmed-Zaki[1]

[1] University of International Business; [2] Kazakh National University

Summary

The paper describes development and comparison tests for high performance visualization of geological models on different platforms. It's motivated by paradigm of ubiquitous computations using distributed systems where any device can potentially act as platform of client interface for the simulator.
Nowadays most of mobile devices have valuable computational resources and thus may be used as platform for scientific computations. So, we present visualization of geological models given in the eclipse format for mobile devices. For rendering we applied standard rasterization pipeline of Vulkan framework with direct coloring of polygons based on numerical values of physical parameters. Such approach leads to necessity of the whole buffer substitution in cases of changing values in new time steps. For this purpose, we implemented double buffering algorithm for vertex buffers used to store color data. To test the performance of visualization comparisons were made based on real time rendering frames per second for geological model differing in cell count.
Also, we implemented and tested geological model visualization performance on advanced desktop stations with leading GPU devices with the implementation of Ray Tracing algorithms. Even though ray tracing algorithm was invented for photorealistic visualization it can show drastic increase of rendering performance due to logarithmic complexity of intersection search algorithms. Because of that it was expected that ray tracing algorithm will show better performance on grid models with high number of cells. In the case of small grids its performance will be lower than performance of rasterization algorithm but its irrelevant due to the overall quickness of such visualization.
Tests shown in the paper mostly confirm that expectations. However, in such comparisons certain additional moments must be considered. Performance of ray tracing algorithm is based on the number of screen fragments or pixels alongside with the number of primitives on the screen. Also, it affects pixel number, or, in other words, percentage of screen, covered by the model. Thus, we show different tests that consider that feature.

Introduction

The development of two-dimensional and three-dimensional data visualization systems for analyzing the results of numerical modeling is one of the developing and important directions. Presentation of results in graphical form improves perception and use by specialists. Using the latest technologies, an application has been developed and tested for visualizing the results of calculations for oil and gas field in real time.

In this field there is a high necessity of convenient data presentation methods, which led to the developed industry of specialized software for visualization of results for numerical computations in various ways. Unlike most other fields of a scientific visualization the visualization in oil and gas field implements classic polygonal render to present a geometry model of the field. Geological models of the oil fields are usually too large in terms of the number of elements to be easily visualized with a proper frame rate. The problems were related to the fact that the computations are carried on the supercomputer systems, while the visualization works on the client device. Recently certain ways of visualization of drastically big amount of polygons were presented [1].

The purpose of this study is a high-quality and continuous visualization of mesh models. In this work, a high-performance mobile and computer application was developed. The use of modern technologies and equipment of graphic processors contributed to increased productivity. In the study, different geomodels were taken as input data for health testing, and a three-dimensional cube model was used as input data for performance testing. The finished program should visualize the results of any numerical mathematical modeling with high performance. The visualization technologies used are described in the next chapter.

Visualization was implemented on computers and mobile devices using Vulkan technology. Today, almost all devices (PCs, mobile phones, tablets) support Vulkan technology. The main advantage of the low-level Vulkan API is direct access to the hardware resources of the GPU and load balancing between the CPU and GPU. Thus, you can achieve high performance and long battery life for mobile devices.
Also, A 3D visualization module using the latest real-time Vulkan Ray Tracing technology has been developed and tested. That visualization module, also visualizes various models in real time, which allows you to visually monitor changes in any process.

Using Vulkan technology, 3D visualization modules have been developed for desktop computers equipped with a discrete graphics card and mobile devices based on the android operating system.

Today, there are several software interfaces that use two-dimensional and three-dimensional computer graphics, such as OpenGL, DirectX and Vulkan. OpenGL is a high-level graphical device programming interface [2, 3, 4]. The OpenGL receiver is the Vulkan graphics API, but they are very different. Vulkan API specifications can be found in the literatures [5, 6, 7, 8]. Vulkan only supports the shader language SPIR-V [9].

Research [10] studies high-performance APIs and makes detailed comparisons between standard OpenGL and the latest APIs such as Vulkan and DirectX 12. The main difference between Vulkan technology and DirectX 12 in cross-platform. DirectX 12 is only available for Windows 10 PCs, while Vulkan is available for almost all devices (PCs, mobile phones, tablets).

In the paper, we develop an application for visualizing oil fields on mobile and desktop platforms using Vulkan technology and conduct a thorough performance analysis of large models' visualization. The paper [11] analyzes the performance of computing on mobile and desktop platforms using the same technology.

In the work [12, 13] developed optimized algorithms for visualizing the results of numerical mathematical modeling on structured grids using the OpenGL shader language library, while our work uses Vulkan technology and a ready-made module visualizes any results of numerical mathematical

modeling on structured and unstructured grids, and to improve performance, we use Vulkan Real Time Ray Tracing Technology [14, 15]. Like in most of those publications, effectiveness of the visualization method is measured by comparing amount of visualized frames per second using different technologies.

Also, need of high performance visualization of the oil-gas field computation results is related to the availability of high performance computations of such problems. Such research is shown on previous work of the authors [16 - 19].

Methods

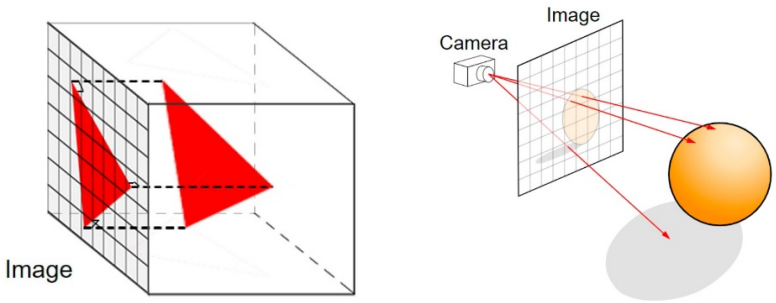
To test and compare several visualization modules, we need to define the method of measuring the effectiveness of the visualization. We tried to achieve smooth transformation manipulations of the geometry model of the oil field. The smoothness of animation is defined by the rate of frames presented on screen per second (FPS). Comfortable and convenient amount of FPS may vary for different people, but in most cases, it must be above 40 in scientific applications. Some users prefer the values above 60. In the paper we will consider, compare and analyze on the base of much higher values of FPS. This is because the further increasing of the geometry model will lead to proportional changes of the FPS amount.

To achieve highest numbers possible to visualize using developed modules we used generated models based on simple cube geometry.

Visualization technologies

There are two methods for converting a 3D model to a two-dimensional image for display on the screen: rasterization of the 3D model and ray tracing. Usually used rasterization method. The method of rasterization is as follows. The primitives of the 3D model are projected onto the plane of the screen and, using special algorithms, the color of each pixel lying inside the projected primitives is obtained.

Ray tracing is a method of creating three-dimensional models using a principle similar to real physical processes. A ray is generated from the camera for each pixel of the screen to the object, then the pixel is colored at the intersection with the object of the ray. An example of rasterization and ray tracing can be seen in Figure 1.



a) b)

Figure 1 *a) rendering method of rasterization, b) rendering method of ray tracing.*

There are the following elements and concepts for working with ray tracing:

- Acceleration structures - a special object that encapsulates the internal picture of the geometry. It can be considered as a type of tree (BVH) that speeds up the search for intersections of the beam and geometry.
- Shader Binding Table (SBT) - a data structure that allows the API to send several shaders (and / or its individual stages) for ray tracing, and then dynamically call the shader from this shader table.

- A new command that starts ray tracing (vkCmdTraceRaysNV).
- A new pipeline that can work with a shader table for ray tracing.

The computer application was implemented using the latest extensions for ray tracing via the Vulkan API

Vulkan is an API for graphic computing devices. Vulkan is a cross-platform API for high-performance graphics access and computing on modern graphics cards. It can be used for heterogeneous devices such as GPUs, mobile devices, and tablets. Vulkan provides applications with direct control over graphics acceleration to improve efficiency and performance. Vulkan is the only high-performance graphical API that works with several operating systems including Windows, Linux, and Android.

The application for visualizations on computers was implemented using the Visual Studio C++programming environment. The Visual Studio programming environment and the NDK Toolkit were also used to develop the mobile application. The input data for visualization are the results of calculations of various problems. They contain the geometry and different characteristics of the reservoir and have the GRDECL format. Figure 2 shows visualization applications for mobile and desktop devices. On mobile devices, this data must be located in a special application folder called "Assets", and special classes and functions of AAssetManager are used to initialize this data.

Reading files using the AAssetManager method is similar to the method for working with files from stdio.h. In the AASSetManager class, AAsset_open is used instead of fopen, aasset_read is used instead of fread, and AAsset_close is used instead of fclose. For initializing files, the assets folder was created and the input data of the model was placed. In applications for reading this data, a dynamic file of the AAsset type is created, and the AASSET_MODE_BUFFER type is assigned. Then the file length is determined (the file length is equal to the number of elements in the file), and a dynamic array of the char type is created with the size of the file length. Then data is read from the file and written to an array with the char type, and the program reads data from this array.

Visualization modules developed using the Vulkan API for mobile devices and desktop systems visualize same models, presented in the same format, as it is shown on the figure 2.

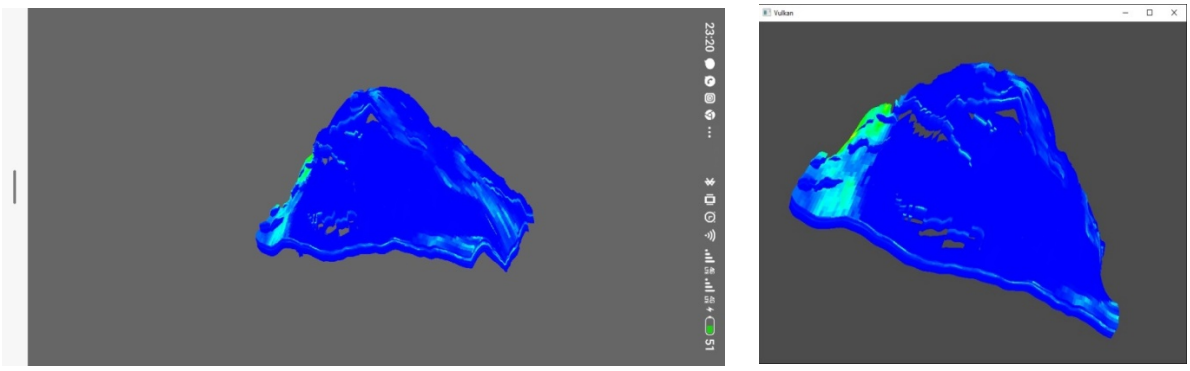


Figure 2 *Application for mobile and desktop devices.*

Test measurements

A personal computer (Core i7 3770 3.40 GHz, 16Gb DDR3) equipped with a discrete graphics card (nVidia GeForce RTX2080 ti, 11Gb GDDR6) was used for performing tests of the visualizer on computers, and for mobile devices - Meizu 16th mobile phone (Qualcomm SnapDragon 845, 6 Gb, 64Gb, Qualcomm Adreno 630). Table 1 describes the characteristics of the devices used for testing.

	Nvidia Geforce 2080 ti	Adreno 630
Number of computing blocks	4352	256
Theoretical performance	28,5/14,2 TFlops	727 GFlops
Memory interface	352 bit	16 bit
Memory Speed	14000 MHz	1866 MHz
Memory bandwidth	448 GB/s	29,9 GB/s
Memory Size	8 ГБ GDDR6	LPDDR4X

Table 1 *Characteristics of GeForce 2080 ti and Adreno 630 graphics devices.*

Comparison of visualization modules for computers and mobile devices. For comparison we used a three dimensional model of a cube with different sizes and filled with random colors shown in figure 3

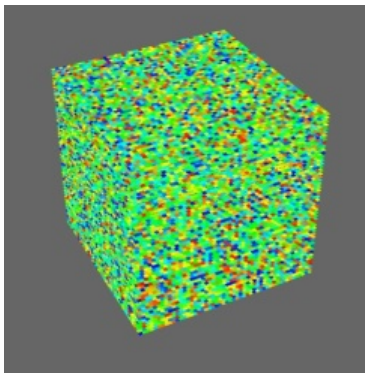


Figure 3 *A three-dimensional model of the cube.*

A comparison of the performance when rendering a three-dimensional model on different platforms with different model sizes is shown in Table 2.

Size	Mobile device	PC
10x10x10	60	4300
20x20x20	60	4500
50x50x50	60	3400
80x80x80	60	1900
100x100x100	45	1370
120x120x120	37	894
140x140x140	24	615

Table 2 *The number of frames per second when rendering a three-dimensional model.*

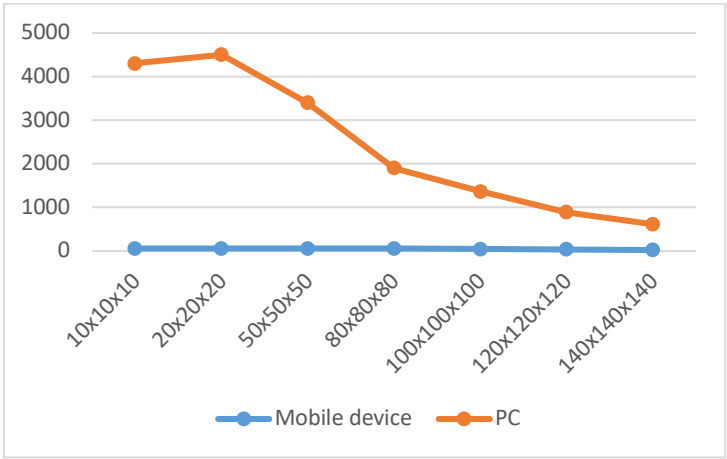


Figure 4 *A graph of the number of frames per second when visualizing an approximate three-dimensional model.*

A comparison of the number of frames per second when visualizing very large two-dimensional grids using Vulkan rasterization and ray tracing is shown in table 3.

Size	Number of grids	Vulkan rasterization (FPS)	Vulkan RT (FPS)
100x100	20,000	3500	2600
500x500	500,000	3600	1285
1000x1000	2,000,000	1650	1095
2000x2000	8,000,000	650	1005
3000x3000	18,000,000	265	585
4000x4000	32,000,000	155	625
5000x5000	50,000,000	100	600
5700x5700	64,980,000	78	610

Table 3 *Number of frames per second when visualizing various large-size grids.*

Each grid cell consists of two triangles, so the number of polygons is twice the number of cells.

As can be seen in table 3, when visualizing using the rasterization method in small sizes, i.e. up to about 2000x2000 or up to 8,000,000 nodes, the number of frames per second is greater than the number of frames per second when visualizing using the ray tracing method. However, as the size of the model increases, the results of visualization using the ray tracing method exceed the results of the rasterization method. The rasterization method shown in the graph in figure 5 shows that the result decreases with increasing size, while the ray tracing method shows a stable result at a size of more than about 18 million nodes.

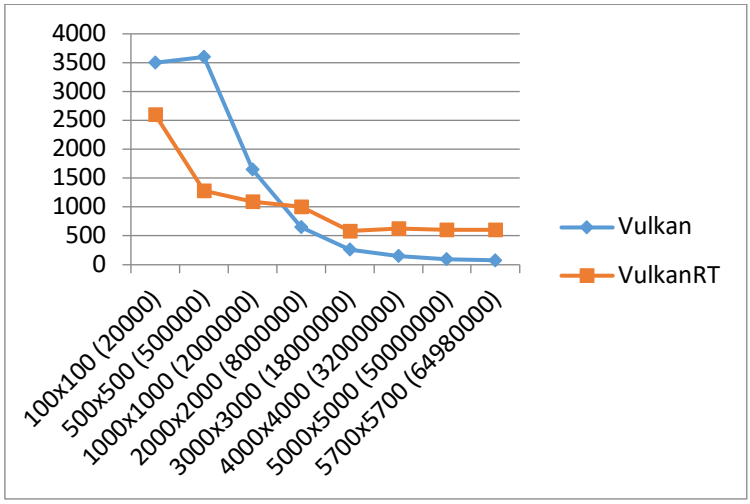


Figure 5 *Relative graph of the number of frames per second when using rasterization and ray tracing methods.*

In addition, the more the grid model occupies the area of the program window, i.e. the closer the model is to the screen, the more it affects performance, and the difference between the two methods changes. The results of visualizing an approximate model using the dimensions in table 3 are shown in table 4.

As can be seen from table 4, the rasterization method works more efficiently than the ray tracing method, up to 4000x4000, but at large sizes, the ray tracing method shows a stable result, and the number of frames per second of the rasterization method decreases. The reason for reducing the result of the rasterization method is that during rasterization, the projection of each node is projected onto the screen plane and pixels are painted over, so as the number of nodes increases, the number of rasterization processes increases. The ray tracing method generates rays through the screen plane, so

the number of frames per second decreases as the model approaches the program window, but the graph in figure 6 shows that the ray tracing method works better with large grids than the rasterization method.

Size	Number of grids	Vulkan rasterization (FPS)	Vulkan RT (FPS)
100x100	20000	3500	2340
500x500	500000	3600	1000
1000x1000	2000000	1660	400
2000x2000	8000000	660	207
3000x3000	18000000	265	166
4000x4000	32000000	157	166
5000x5000	50000000	102	149
5700x5700	64980000	77	200

Table 4 *The number of frames per second when visualizing various large grids of an approximate model.*

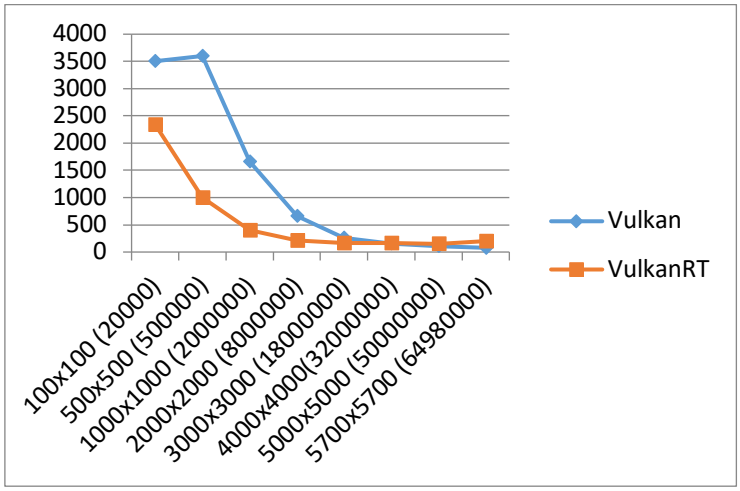


Figure 6 *Graph of the number of frames per second when visualizing various large grids of an approximate model.*

In addition, the dimension of the grid model affects performance, and the difference between the two methods also changes. Table 5 shows a comparison of the number of frames per second of the two methods for visualizing three-dimensional grids. The cube model was used for comparison.

Size	Number of grids	Vulkan rasterization (FPS)	Vulkan RT (FPS)
10x10x10	2000	3400	2450
20x20x20	16000	3600	2350
50x50x50	250000	3300	2240
100x100x100	2000000	1460	1700
150x150x150	6750000	515	1550
200x200x200	16000000	255	1325
250x250x250	31250000	146	1120
280x280x280	43904000	109	1000
300x300x300	54000000	88	900

Table 5 *The number of frames per second when visualizing various three-dimensional grid models.*

As can be seen from table 5, the ray tracing method shows several times higher performance than the rasterization method when rendering models larger than 100x100x100. Since in the case of rasterization

all the nodes of the model are taken into account, and in the case of ray tracing, the ray returns only the color at the intersection, and the invisible layers behind are not taken into account, therefore, for three-dimensional models, the ray tracing method gives good performance. The results of table 4 are shown in the graph in figure 7.

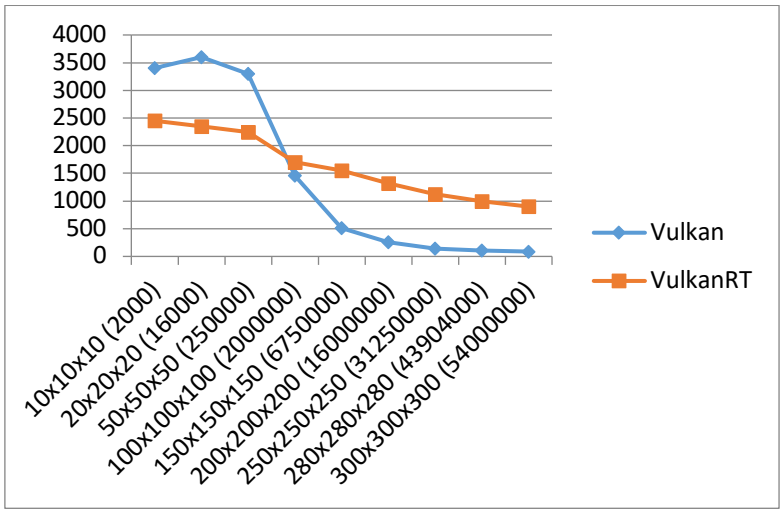


Figure 7 *Graph of the number of frames per second when visualizing a three-dimensional grid model.*

The results of visualization of an approximate three-dimensional model are shown in Table 6 and on the graph in figure 8.

Size	Number of grids	Vulkan rasterization (FPS)	Vulkan RT (FPS)
10x10x10	2000	3350	2160
20x20x20	16000	3500	2200
50x50x50	250000	1840	1980
100x100x100	2000000	770	1690
150x150x150	6750000	358	1410
200x200x200	16000000	200	1270
250x250x250	31250000	120	1230
280x280x280	43904000	88	1130
300x300x300	54000000	75	1060

Table 6 *The number of frames per second when visualizing various large grids of an approximate three-dimensional model.*

Analysis and discussion

Analyzing the results obtained when comparing visualization modules on mobile and desktop devices in Table 2, we can see that the mobile device limits the number of frames per second when rendering the model to a size of 80x80x80, and FPS is 60. When visualizing a model larger than 80x80x80 cells, the number of FPS is reduced to 24. The results obtained on the desktop GPU are very high and starts from 4300 FPS when working with small-size models and decreases to 615 FPS as the size increases.

Table 3 shows the results of FPS when rendering a two-dimensional model on a desktop GPU using the rasterization and ray tracing method. Visualization by rasterization method to the size 2000x2000 shows 1650 frames per second, and in the ray tracing method, the FPS is equal to 1095. As the model size increases, the fps of ray tracing rendering exceeds the results of the rasterization method. In table 4, you can see the FPS results of an approximate two-dimensional model. When drawing an approximate model to a size of 3000x3000, you can see that the rasterization method shows more FPS than the ray

tracing method, but with large cell sizes, the number of frames per second when rendering by the rasterization method is lower than the number of frames per second when rendering by the ray tracing method.

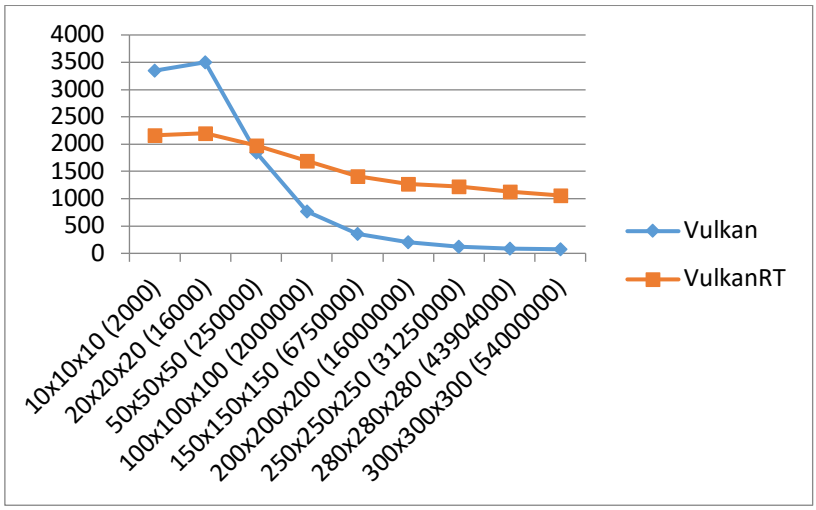


Figure 8 *Graph of the number of frames per second when visualizing various large grids of an approximate three-dimensional model.*

Table 5 shows the results of FPS when rendering a three-dimensional model on a desktop GPU using the rasterization and ray tracing method. You can see from the table that rendering using ray tracing shows a higher FPS than the rasterization method when rendering a model larger than 100x100x100. Table 6 shows the FPS results of an approximate three-dimensional model. Here you can see that starting from the size 50x50x50 of the three-dimensional model, the number of frames per second when rendering using ray tracing is several times greater than the number of frames per second when rendering by rasterization.

When visualizing three-dimensional models, using a desktop GPU is more efficient than using a mobile device. Because when comparing the results of these two devices, the desktop device showed a higher FPS, which means that the desktop device is more useful for visualizing very large geological models. But the mobile device, despite the low performance, showed not a bad result for such platforms.

As a result of comparison, the rasterization method showed good performance at small sizes, but as the size of the grid model increases, the ray tracing method showed stable and high performance. However, as you approach the model, you can see that the number of frames per second varies significantly in the ray tracing method. This is because the area of the desired ray increases, and the rasterization method does not change the number of operations, so it does not change significantly, since each cell is projected onto the screen. In addition, when visualizing three-dimensional grid models, the ray tracing method showed higher performance than the rasterization method. This is because the rasterization method calculates the projection of each cell on the plane of the screen, and the ray tracing method does not consider the invisible parts of the object.

Such behavior is mostly expected, because those two methods have different algorithmic complexity. Let's temporary ignore the fact of its parallelization to the big amount of threads. Then rasterization algorithm acts as brute force algorithm, and linearly processes all the polygons each frame. While the algorithm of ray tracing processes only goes from the root of the spatial tree of polygons to its leaves, which means it has logarithmic complexity. That logarithm is multiplied to the amount of rays that can be considered as constant, because it is related only to the screen size. Thus we compare two functions: linear for rasterization, and logarithmic with some constant multiplied in the case of ray tracing. The fact that from some point logarithmic function will always be lesser than the other function is obvious.

However, it is necessary to show experimentally, that the mentioned point can be achieved in practice. Such experiments are shown in this paper. The fact of GPU parallelization does not critically affect to this comparison due to the fact that system automatically launches most possible amount of threads. Thus in both algorithms there are similar parallelization effectiveness.

Therefore, for visualization of three-dimensional large grid models, it is advantageous to use the ray tracing method.

Conclusion

This paper describes the work on developing a high-performance mobile and computer application. The results of a mobile application using the Vulkan API and a computer application using the raserization method and ray tracing method are obtained. A different-sized cube model was used as input data for performance testing. Comparative analyses of the results of visualization modules for mobile and desktop devices, and comparative analyses of the results of a computer application using two methods were performed. Comparing of results and further analysis show that using ray tracing algorithms may be effective in cases of visualization of large amount of polygons/cells in geological model. Mobile devices does show efficiency of visualization that is enough to use it as client side application with geological models with several millions of polygons.

Acknowledgments

This research was funded by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan (Grant No. BR05236447).

References

[1] A first look at Unreal Engine 5, 2020 - https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5
[2] Mark S., Kurt A. “The OpenGL Graphics System: A Specification.” (2008).
[3] Sellers G., Wright R.S. Jr., Haemel N. OpenGL SuperBible: Comprehensive Tutorial and Reference (6th Edi- tion). Addison-Wesley Professional; 6 edition (July 31, 2013) 2013 P. 848.
[4] OpenGL programming guide : the official guide to learning OpenGL, version 4.3 / Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane ; the Khronos OpenGL ARB Working Group.---Eighth edition.
[5] Khronos Group. Vulkan. https://www.khronos.org/vulkan/. Accessed: 2017-11-13.
[6] Sellers G., Kessenich J. Vulkan Programming Guide: The Official Guide to Learning Vulkan. Addison-Wesley Professional; 01 edition (31 Oct. 2016), p. 478.
[7] Khronos Vulkan Working Group. Vulkan 1.0.98 - A Specification (with KHR extensions). 1.0.98. Jan. 2019.
[8] Pawel Lapinski, Vulkan Coockbook. Work through recipes to unlock the full potential of the next generation graphics API—Vulkan. Packt Publishing Ltd. Birmingham (2017).
[9] Khronos Group. SPIR Overview. https : / / www . khronos . org / spir/. Accessed: 2018-03-03.
[10] Shiraef, Joseph A. “An exploratory study of high performance graphics application programming interfaces.” (2016).
[11] N. Mammeri and B. Juurlink, "VComputeBench: A Vulkan Benchmark Suite for GPGPU on Mobile and Embedded GPUs," 2018 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, NC, 2018, pp. 25-35
[12] Badretdinov M. R. , Badretdinov T. R. , Borshchuk M. S., Primeneniye biblioteki opengl dlya vizualizatsii rezul'tatov chislennogo matematicheskogo modelirovaniya na setkakh bol'shoy razmernosti [Using the opengl library to visualize the results of numerical mathematical modeling on large-dimensional grids], Vestnik UGATU, 2015. № 4. 84-94
[13] Abraham, Frederico & Celes, Waldemar. (2009). Distributed Visualization of Complex Black Oil Reservoir Models.. 87-94. 10.2312/EGPGV/EGPGV09/087-094.

[14] Khronos Group. Ray Tracing In Vulkan. – https://www.khronos.org/blog/ray-tracing-in-vulkan#raytracing1a Accessed: 2020-03-25.
[15] NVIDIA Vulkan Ray Tracing Tutorial – https://developer.nvidia.com/rtx/raytracing/vkray Accessed: 15.02.2019.
[16] D.Zh. Akhmed-Zaki, T.S. Imankulov, B. Matkerim, B.S. Daribayev, K.A. Aidarov, O.N. Turar. Large-scale simulation of oil recovery by surfactant-polymer flooding. Eurasian Journal of mathematical and computer applications. – 2016. Volume 4, Issue 1. – P. 12-31.
[17] Akhmed-Zaki D.Zh., Daribayev B.S., Imankulov T.S., Turar O.N. High-performance computing of oil recovery problem on a mobile platform using CUDA technology. Eurasian Journal of mathematical and computer applications. – 2017. Volume 5, Issue 2. – P. 4-13.
[18] T.S. Imankulov, D.Zh. Akhmed-Zaki, B.S. Daribayev and O.N. Turar. HPC Mobile Platform for Solving Oil Recovery Problem. Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016), Volume 2 Lisbon, Portugal. 29 - 31 July 2016. – P. 595-598.
[19] Akhmed-Zaki D., Danaev N., Mukhambetzhanov S., Imankulov T. Analysis and Evaluation of Heat and Mass Transfer Processes in Porous Media Based on Darcy-Stefan's Model. ECMOR XIII - 13th European Conference on the Mathematics of Oil Recovery. 2012
[20] Klochkov M. A. "K resheniyu zadachi vizualizatsii rezul'tatov modelirovaniya protsessov razrabotki neftegazovykh mestorozhdeniy" ["Towards a solution to the problem of visualizing the results of modeling oil and gas field development processes"], Proceedings of the Institute of Mathematics and Computer Science, Udmurt State University, vol. 49, 2017, pp. 3-16.
[21] E.A. Gladkov. Geologicheskoye i gidrodinamicheskoye modelirovaniye mestorozhdeniy nefti i gaza: uchebnoye posobiye [Geological and hydrodynamic modeling of oil and gas fields: a training manual]. Tomsk Polytechnic University. - Tomsk: Publishing House of Tomsk Polytechnic University, 2012. - 99 p.
[22] Mullen, Tim R., Christian Kothe, Yu M. Chi, Alejandro Ojeda, Trevor Kerth, Scott Makeig, Gert Cauwenberghs and Tzyy-Ping Jung. “Real-time modeling and 3D visualization of source dynamics and connectivity using wearable EEG.” 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (2013): 2184-2187.