Compact GPU-based Visualization Method for High-resolution Resulting Data of Unstable Oil Displacement Simulation

Timokhin Petr[1], Mikhaylyuk Mikhail[1]
webpismo@yahoo.de | mix@niisi.ras.ru
[1]Federal State Institution «Scientific Research Institute for System Analysis of the Russian Academy of Sciences», Moscow, Russia

In the paper the task of real-time synthesis of quality images of resulting data obtained in simulation of unstable oil displacement from porous media is considered. A new, GPU-based method to construct and visualize on UltraHD screens a polygonal model of the isosurface of the saturation of displacing liquid was proposed. The method is based on distributing and parallelizing of «marching cubes» threads between GPU cores by means of programmable tessellation. As initial graphic primitives, quadrangular parametric patches are used, the processing of which on the GPU is high-performance and has low video memory overhead. The proposed method was implemented in visualization software and successfully tested. The proposed solution can be used in researches in oil and gas industry as well as in virtual environment systems, virtual laboratories, scientific and educational applications, etc.

Keywords: *visualization, real-time, oil displacement, isosurface, marching cubes, GPU, tessellation.*

1. Introduction

With the increasing complexity of hydrocarbon extraction, a high priority is acquired by digital technologies aimed at raising oil recovery of oil-bearing reservoirs [3]. In particular, technologies based on computer simulation and visualization of the processes of unstable oil displacement from porous media by water and polymer solutions are widely demanded [6]. First part (simulation) is very important and concerns numerical solution of the hydrodynamic task. Depending on the initial conditions and the given accuracy, the computing of the task can last for several days. This work focuses on the second important part of the problem – the development of methods for real-time 3D visualization (at least 25 frames per second) of numerical results of oil displacement simulation, providing UltraHD quality. Compared to analysis of numerical data, researching 3D representations of numerical simulation results allows developed computing models to be evaluated and validated at a qualitatively higher level, and target scientific and practical knowledges to be faster extracted.

One of the keys is the visualization of the isosurface of the saturation of the displacing liquid – the surface of a constant value of the 3D saturation field obtained at each step of the simulation. There are two main approaches to visualize the isosurface. One is to cast rays from a viewpoint through screen pixels till the intersection with the isosurface [7]. Another is to construct a polygonal model of the isosurface in the cells of a 3D grid (render grid) from typical sets of triangles («marching cubes», MC) [4,5]. The former approach allows to obtain high-quality images of isosurface, however, the rate of image synthesis significantly falls on UltraHD screens, which opposes the isosurface to be rendered in real-time. In the latter approach, the quality of synthesized images is determined by the detail of the render grid: the higher it is, the less angularity the isosurface model is subjected to. This is clearly visible in UltraHD. Therefore, to obtain high-quality images in real-time, effective methods to construct polygonal isosurface model on detailed render grid using distributed computing [1] and calculation parallelization on multi-core GPU, are needed.

The paper proposes a new, distributed method which performs computations of «marching cubes» in parallel GPU-threads generated by means of programmable tessellation of quadrangular graphic primitives [2]. The solution proposed is implemented using C++, GLSL languages and OpenGL library.

2. Constructing a polygonal model of the isosurface

Consider the task of visualization of a polygonal model of the surface of some constant value S^* of the saturation field of the displacing fluid. This field is defined by a render grid R of size $m \times n \times q$ cells. Each of 8 vertices of the cell corresponds to a certain value S of the saturation of the displacing fluid. We will construct polygons only in those cells (cubes) of the grid R, for all 8 vertices of which is not met the same condition: $S < S^*$ or $S \geq S^*$. In other words, only the cells, that do not lie completely inside or outside the isosurface, will be affected.

Consider some $R_{i,j,k}$ cell. Let us number the vertices of this cell from 0 to 7, and the edges – from 0 to 11. We mark every vertex of the cell, for which $S < S^*$, with a bit value equal to 1, otherwise – 0. Denote by K the configuration of 8 written one after another bits. To every K value (from 0 to 255) a set of triangles (polygons), which will be a part of the polygonal model of the isosurface (containing in the render grid cell), is uniquely corresponded. Every such set includes from 0 to 5 triangles. The vertices of the triangle lie on the edges of the cell, so that one edge contains no more than one vertex. The set of triangles is given by a sequence of 16 indices: 3 edge indices per triangle and one index (-1) of the end of the sequence. All the 256 sequences form a table T of sets of triangles (see [4]). For the vertex specified by some edge index e in table T the coordinates P are calculated as follows

$$P = P_A + \frac{S^* - S(A)}{S(B) - S(A)}(P_B - P_A), \qquad (1)$$

where A, B are vertices of eth edge of the cell, P_A, P_B are their coordinates, and $S(A)$, $S(B)$ are values of saturation of displacing liquid at vertices A and B.

Let us denote by $W_{i,j,k}$ a GPU-thread constructing a part of the polygonal model of the isosurface in the $R_{i,j,k}$th cell (hereinafter referred to as «marching cubes» thread). Then, the entire polygonal model of the isosurface can be constructed by creating and performing the «marching cubes» threads for all the cells of the render grid.

3. Tessellation based «marching cubes»

In this paper, it is proposed to create «marching cubes» threads by means of programmable tessellation (subdividing) of patch-quads – quadrangular parametric graphic primitives (hereinafter referred to as *patches*). Having programmed the graphics pipeline of the GPU in a certain way, one patch can be subdivided into a regular grid of size up to $D \times D$ vertices. Here $D = L_{max} + 1$ and L_{max} is the *maximum tessellation level* – the largest number of segments, the patch side can be subdivided into (at least 64 in OpenGL 4.0). According to the pipeline architecture each vertex is processed in a separate graphic thread which we program to construct the corresponding part of the polygonal isosurface model. The main advantage of the approach proposed is that the vertices (threads) are generated

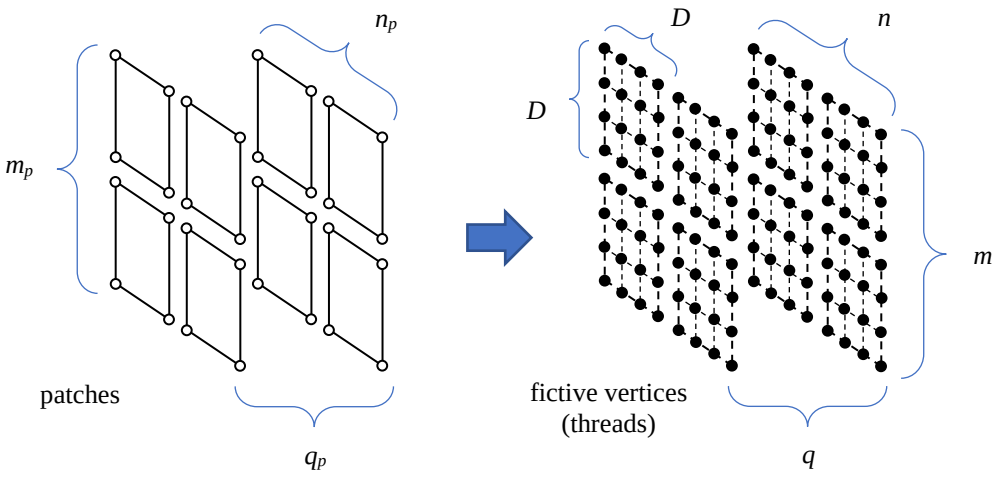

Fig. 1. Generating «marching cubes» threads.

directly in GPU, preserving video memory resource and without leaving graphics pipeline, in contrast to CUDA-based solutions.

To obtain $m \times n \times q$ «marching cubes» threads, a 3D array of patches, consisting of $m_p = \lceil m/D \rceil$ rows, $n_p = \lceil n/D \rceil$ columns and $q_p = q$ slices, is created (see Figure 1). Given array is sent to the graphics pipeline every visualization frame, where patches are distributed between the GPU cores and processed by means of shader programs developed. These are tessellation control shader, tessellation evaluation shader, geometry and fragment shaders.

Tessellation control shader (TC). Here the computation of two sets of parameters is executed.

The first set includes parameters specifying the width and the height of 2D grid of vertices (a group of «marching cubes» threads). The width and the height of thread group are determined by tessellation levels l_w and l_h of the patch, which are calculated as follows

$$l_w = max\big(min\big(L_{max}, n - Dj_p - 1\big), 1\big),$$
$$l_h = max\big(min\big(L_{max}, m - Di_p - 1\big), 1\big).$$

The second set includes three indices (i_p, j_p, k_p) of the row, column and slice of the patch in 3D array, required further for setting correspondence of created threads to the cells of the render grid. The calculation of the indices (i_p, j_p, k_p) is based on built-in variable of TC-shader, called $gl\_PrimitiveID$ which is the running number $g \in [0, m_p n_p q_p - 1]$ of passing patch, and is performed as follows

$$k_p = \left\lfloor \frac{g}{N} \right\rfloor, \quad i_p = \left\lfloor \frac{g - Nk_p}{M} \right\rfloor, \quad j_p = g - Nk_p - Mi_p,$$

where $N = m_p n_p$ and $M = Nq_p$.

Tessellation evaluation shader (TE). This shader performs processing of every vertex, obtained as a result of patch tessellation, in a parallel thread, and sets the correspondence of the vertex (thread) to a render grid cell. The latter is implemented by adding to the vertex the attributes (i_c, j_c, k_c) – indices of the row, column and slice of the corresponding cell in render grid, which are calculated as follows

$$i_c = Di_p + i, \quad j_c = Dj_p + j, \quad k = k_p,$$

where i and j are indices of the row and column of the vertex in 2D grid of vertices derived after tessellation the patch:

$$i = \lfloor l_h v + \varepsilon \rfloor, \quad j = \lfloor l_w u + \varepsilon \rfloor,$$

where $(u, v) \in [0,1]$ are normalized real coordinates of the vertex in 2D grid, provided by TE-shader, and ε is a small constant which compensates machine error of real numbers representation.

Geometry shader. This shader processes every (i_c, j_c, k_c)th vertex, which results either in replacing the vertex by a constructed part of the polygonal model of the isosurface, or in discarding the vertex without any geometry creation. The processing starts with calculating of the number K of predefined set of triangles (see Section 2):

$$K = \sum_{p=0}^{7} \left(2^p \left(S(V_p) < S^*\right)\right),$$

where p is running number of a vertex in the $(i_c, j_c, k_c)th$ cell of render grid, and $S(V_p)$ is the value of saturation of displacing liquid at vertex V_p of the cell. If $K = 0$ or $K = 255$ (the cell completely lies inside or outside the isosurface), then any triangles are not created in the cell (geometry shader simply exits). Otherwise, according to the table T (see Section 2), the set of triangles, corresponding to K, is determined. Positions of vertices of these triangles are calculated using Eq. (1), and then triangles are emitted (see [2] for more detail). Note, that before starting the visualization, the saturation field of the displacing fluid is loaded into video memory as floating-point 3D-texture, and the table T – as an integer 2D-texture.

The triangles produced by the geometry shader are rasterized (fixed function of the pipeline), resulting into pixels (fragments) forming the image of the isosurface model. Pixel colors are calculated in parallel, independently of each other by means of developed **fragment shader** based on the Phong illumination model [2] with directional light source. Figure 2 shows general scheme of the main visualization pipeline, representing the whole workflow from simulated data till screen image.

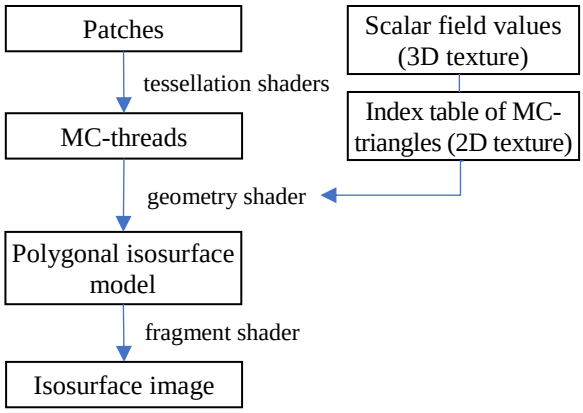

Fig. 2. Main visualization pipeline.

4. Results

Based on the method proposed, a software to visualize results of simulation of unstable oil displacement from porous

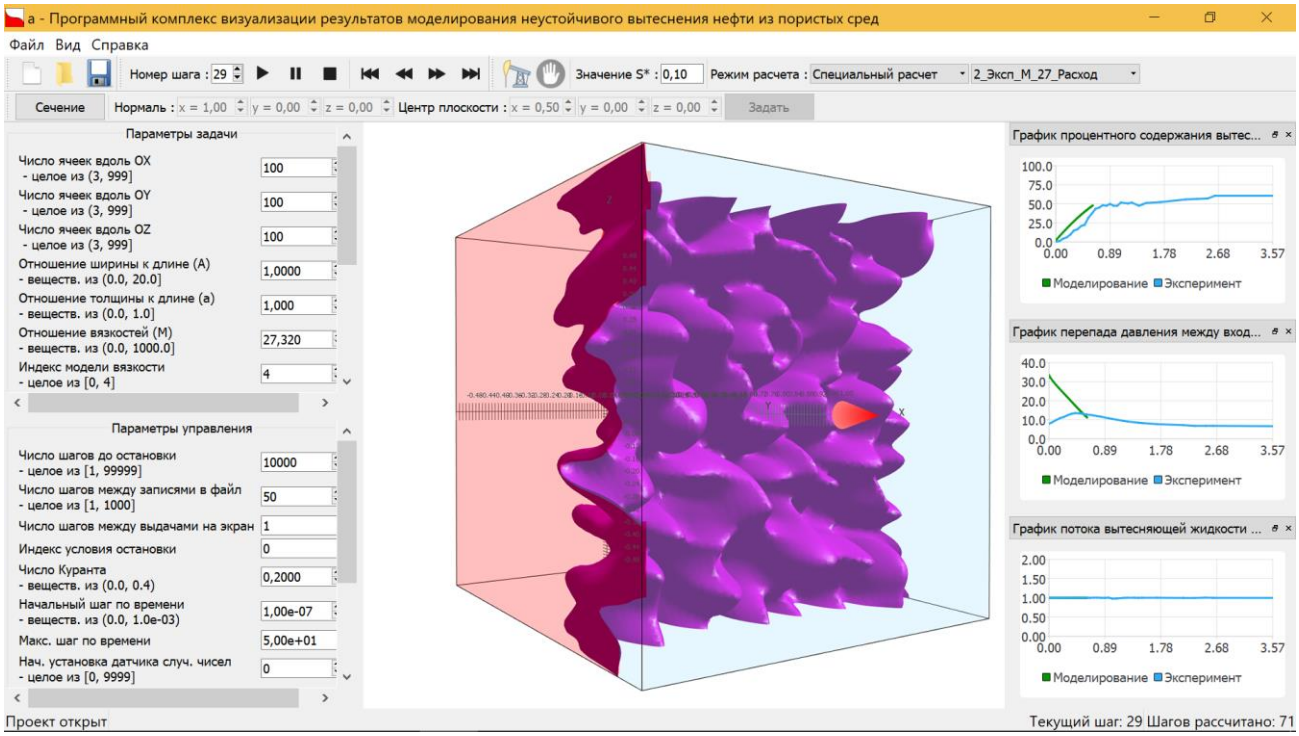

Fig. 3. Developed visualizer of data obtained in simulation of unstable oil displacement from porous media.

media was created. Using the developed solution, a research of about 70 steps of simulation of displacing oil by water on a calculation grid of 100^3 points was carried out. Figure 3 shows the change of the isosurface of water saturation, corresponding to the moment of breakthrough of displacement front. On the figure one can see the characteristic tongues of the displacing liquid (called «fingers»). The polygonal model of the isosurface was constructed and visualized at screen resolution of 3840x2160 pixels using GeForce GTX 1080 Ti graphics card, the average visualization rate was about 100 frames per second.

5. Conclusions

The paper proposes a new, distributed method to construct and visualize on the GPU a polygonal model of the isosurface, which provides real-time synthesis of quality images of the isosurface on UltraHD screens. The proposed solution is based on creation and execution of «marching cubes» threads on the GPU using the developed set of tessellation, geometry and fragment shaders. The novelty of the work consists in distribution and parallelization of the «marching cubes» threads between GPU cores by means of programmable tessellation of quadrangular parametric graphic primitives – patches. Their processing is high-performance due to hardware support at the GPU architecture level and has low video memory overhead.

The proposed method was implemented in visualization software, and it was tested on data obtained in simulation of unstable oil displacement by water. Approbation of the software has confirmed that the methods and algorithms created meet the requirements for visualization of the results of simulation of unstable oil displacement. The developed solution can be used in researches in oil&gas industry, as well to build virtual environment systems, virtual laboratories, scientific visualization systems, etc.

6. Acknowledgements

This research was supported by the Russian Foundation for Basic Research (project No. 16-29-15099).

7. References

[1] Akayev A.A, Kuzin A.K., Orlov S.G., Chetverushkin B.N., Shabrov N.N., Iakobovski M.V. Generation of Isosurface on a Large Mesh. In *Proceedings of the IASTED International Conference on Automation, Control, and Information Technology (ACIT 2010)*, pages 236–240. ACTA press, 2010.

[2] Bailey M., Cunningham S. *Graphics Shaders: Theory and Practice, Second Edition*. CRC Press, 2011.

[3] Betelin V.B., Smirnov N.N. About the problem of import independence in the oil and gas industry. Computational simulation of active impacts on oil reservoirs. In *Proceedings of the V International Conference «Mathematics and information technologies in the oil and gas complex»*, pages 8–45. SurGU, 2017.

[4] Bourke P. Polygonising a scalar field. URL: http://paul-bourke.net/ geometry/polygonise/ (review date: 15.05.2019).

[5] Matsumura M., Anjo K. Accelerated isosurface polygonization for dynamic volume data using programmable graphics hardware. In *Proceedings of SPIE-IS&T Electronic Imaging, Visualization and Data Analysis*, 5009:145-152, 2003.

[6] Mikhaylyuk M.V., Timokhin P.Yu., Maltsev A.V., Nikitin V.F., Skryleva E.I., Tyurenkova V.V. Modeling and visualization of process of oil displacement from porous medium. *Proceedings in Cybernetics International Journal*, 3(23):35–41, 2016.

[7] Parker S., Shirley P., Livnat Y. and et. al. Interactive Ray Tracing for Isosurface Rendering. In *Proceedings of the IEEE Visualization 98 (VIZ'98)*, pages 233-238, 1998.