ПРОГРАММИРОВАНИЕ, 2020, № 3, с. 66–72

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.94

МЕТОД ИЗВЛЕЧЕНИЯ ПОВЕРХНОСТЕЙ УРОВНЯ НА GPU С ПОМОЩЬЮ ПРОГРАММИРУЕМОЙ ТЕССЕЛЯЦИИ

© 2020 г. П. Ю. Тимохин[a],*, М. В. Михайлюк[a],**

[a] *ФГУ "ФНЦ Научно-исследовательский институт системных исследований РАН"*
117218 Москва, Нахимовский пр., 36, к. 1, Россия

**E-mail: webpismo@yahoo.de*

***E-mail: mix@niisi.ras.ru*

Поступила в редакцию 20.12.2019 г.
После доработки 10.01.2020 г.
Принята к публикации 18.01.2020 г.

В статье рассматривается задача визуализации в масштабе реального времени детализированных трехмерных скалярных полей с помощью поверхностей уровня – поверхностей постоянного значения скалярного поля. Предлагается новый метод обхода ограничений, связанных с интенсивным считыванием фиктивных вершин из видеопамяти и накладными затратами на их хранение, которые возникают при реализации на GPU-методов полигонизации поверхностей уровня. Предлагаемый метод основан на эффективном создании GPU-потоков, в которых выполняется построение модели поверхности уровня, путем запрограммированной тесселяции четырехугольных патчей на регулярные сетки вершин. В работе предлагается модифицированная технология реализации на GPU "шагающих кубиков", основанная на разработанном методе, которая позволяет значительно сократить временные затраты на создание GPU-потоков и накладные затраты видеопамяти. На основе созданных решений был реализован программный комплекс построения и визуализации полигональных моделей поверхностей уровня, проведена его апробация и верификация синтезированных изображений.

DOI: 10.31857/S0132347420030103

1. ВВЕДЕНИЕ

В настоящее время во многих важных научных и прикладных областях востребована визуализация трехмерных скалярных полей, полученных в результате численного моделирования или сканирования объектов (процессов) со сложной внутренней структурой. Одним из эффективных методов является визуализация в масштабе реального времени [1] (с частотой синтеза изображений не менее 25 раз в секунду) поверхности постоянного значения скалярного поля (*поверхности уровня*). Примерами являются задачи визуализации костного скелета в медицинской компьютерной томографии [2], поверхности раздела фаз в цифровой модели керна [3], трехмерной модели рельефа с отрицательными уклонами в системах виртуального окружения [4] и др. Извлечение поверхности уровня из поля основано на вычислении в ячейках скалярной сетки точек с заданным постоянным значением поля. Чем сложнее объект исследования, тем более детальная скалярная сетка требуется для его адекватного представления, и, соответственно, тем больше вычислений необходимо произвести, чтобы извлечь поверхность уровня. В современных условиях отображения результатов визуализации на экранах сверхвысокой четкости и стереоэкранах [5] объем и время таких графических вычислений возрастают в разы, что препятствует синтезу изображений поверхности уровня в масштабе реального времени. В этой связи возникает задача создания эффективных методов и алгоритмов извлечения поверхностей уровня, основанных на распараллеливании вычислений на современных тысячеядерных графических процессорах (GPU) и использовании технологий параллельной обработки графических данных.

В данной работе предлагается новый метод решения этой задачи на GPU, основанный на полигонизации скалярного поля (извлечении набора треугольных граней, приближающего поверхность уровня), которая выполняется в масштабе реального времени в параллельных GPU-потоках, созданных с помощью программируемой тесселяции параметрических графических примитивов [6]. Предложенное решение реализовано на языке C++, языке GLSL программирования шейдеров и с использованием графической библиотеки OpenGL.

2. ПОДХОДЫ К ИЗВЛЕЧЕНИЮ ПОВЕРХНОСТЕЙ УРОВНЯ

Можно выделить два основных направления развития методов извлечения поверхностей уровня.

К первому направлению принадлежат методы, основанные на испускании лучей (“бросании лучей”, ray casting) из позиции наблюдателя через пикселы экрана до пересечения с поверхностью уровня [7–9]. Данный подход позволяет получать изображения поверхностей уровня высокого качества, однако скорость синтеза таких изображений существенно падает с ростом числа обрабатываемых пикселов (просчитываемых лучей), что препятствует визуализации в реальном времени на экранах со сверхвысоким разрешением. Последние исследования [10] показывают, что появившаяся недавно аппаратная поддержка трассировки лучей у видеокарт NVidia RTX может существенно улучшить данную ситуацию.

Ко второму направлению относятся методы, основанные на построении полигональной аппроксимации поверхности уровня в ячейках трехмерной сетки (рендер-сетки), совмещенной с сеткой скалярного поля [11–19]. При данном подходе находятся *активные* (пересекаемые поверхностью уровня) ребра и ячейки рендер-сетки, для которых по определенным правилам создаются полигоны поверхности уровня. Выделяют прямые [11–13], двойственные [14–16] и гибридные [17–19] методы полигонизации поверхности уровня. В прямых методах внутри каждой активной ячейки создаются полигоны, вершины которых лежат на активных ребрах ячейки (классическим примером является метод “шагающих кубиков” [11]). В двойственных методах полигоны создаются для активных ребер, а вершины этих полигонов создаются внутри активных ячеек. Гибридные методы являются комбинацией прямых и двойственных методов и ведут себя как двойственные методы в случаях, когда активные ячейки содержат особенности поверхности уровня, такие как острые ребра и конические вершины. Обширное исследование и сравнение методов полигонизации поверхностей уровня представлено в работе [20].

С добавлением в графический конвейер GPU программируемых (шейдерных) стадий исследования стали активно развиваться в направлении распараллеливания на GPU обработки активных ячеек и ребер [21–27]. Хорошие результаты были получены при распараллеливании прямых методов, в которых, в отличие от двойственных методов, каждая активная ячейка триангулируется независимо от других. Эффективность GPU-версий прямых методов сильно зависит от временных затрат на создание и исполнение GPU-потоков и расхода видеопамяти. В эпоху тысячеядерных GPU с иерархической структурой видеопамяти узким местом визуализации высокополигональных моделей является число обращений к глобальной видеопамяти (самой большой и медленной части видеопамяти). Снизить число таких обращений можно, выполняя построение полигональной модели для активной ячейки в геометрическом шейдере – одной из программируемых стадий графического конвейера. Чтобы запустить один GPU-поток с геометрическим шейдером, достаточно подать на его вход одну вершину. Распространенный подход состоит в создании в видеопамяти вершинного буфера (VBO) с фиктивными вершинами (по вершине на каждую ячейку рендер-сетки) и отправке этого массива вершин на графический конвейер. Фиктивные вершины поступают на вычислительные ядра GPU, где на каждом ядре параллельно, независимо друг от друга геометрический шейдер заменяет вершину на определенную полигональную конструкцию, если ячейка является активной. И хотя построение полигональной модели поверхности уровня фактически выполняется внутри GPU, в данном подходе имеются существенные ограничения, связанные с интенсивным считыванием вершинных данных из глобальной видеопамяти и дополнительными накладными затратами на их хранение. Эти ограничения препятствуют увеличению размеров рендер-сетки и построению в масштабе реального времени поверхностей уровня для сложных объектов.

В данной работе предлагается новый метод обхода описанных ограничений, основанный на задействовании в процессе построения полигональной модели поверхности уровня двух новых программируемых стадий – шейдера управления тесселяцией и шейдера вычисления тесселяции [6]. Данные стадии выполняются перед геометрическим шейдером и в общем случае предназначены для параллельной генерации на GPU большого числа связанных треугольных аппроксимаций из параметрических графических примитивов (патчей), как например, в задаче построения полигональной модели земной поверхности [28]. Данная работа основана на использовании одного из частных случаев программируемой тесселяции, при котором из четырехугольного патча (патча-квада) на GPU генерируется регулярная сетка вершин. В исследовании показано, как с помощью этой возможности можно существенно сократить время создания GPU-потоков и накладные затраты видеопамяти на примере задачи построения полигональной модели поверхности уровня с помощью “шагающих кубиков”.

3. ТЕХНОЛОГИЯ РЕАЛИЗАЦИИ НА GPU “ШАГАЮЩИХ КУБИКОВ” С ПОМОЩЬЮ ТЕССЕЛЯЦИОННЫХ ШЕЙДЕРОВ

Пусть имеется рендер-сетка R размера $m \times n \times q$ ячеек, каждому узлу (вершине) которой соответствует некоторое значение S исследуемого скалярного поля. Рассмотрим задачу построения в активных ячейках сетки R полигональной модели поверхности некоторого постоянного значения S^*. Активной будем считать каждую ячейку, для всех восьми вершин которой не выполняется условие $S < S^*$ или $S \geq S^*$, т.е. исключаются ячейки, которые лежат полностью внутри или снаружи поверхности уровня.

Рассмотрим некоторую $R_{i,\,j,\,k}$-ю ячейку. Пронумеруем вершины этой ячейки целыми числами от 0 до 7, а ребра – от 0 до 11. Запишем единичный бит для каждой вершины, у которой $S < S^*$, в противном случае – нулевой (см. рис. 1). Согласно правилу “шагающих кубиков” на ребро, вершины которого имеют разные биты (активное ребро), помещается вершина треугольника (полигона), при этом координаты P этой вершины находятся как

$$P = P_a + \frac{S^* - S_a}{S_b - S_a}(P_b - P_a), \qquad (1)$$

где P_a и P_b – координаты вершин активного ребра, а S_a и S_b – значения исследуемого скалярного поля в вершинах активного ребра. Обозначим через K конфигурацию из восьми записанных подряд полученных битов. Каждому значению K (от 0 до 255) однозначно соответствует список треугольников, который будет частью полигональной модели поверхности уровня. Каждый такой список включает в себя от 0 до 5 треугольников и задается последовательностью из 16 индексов: на каждый треугольник по три индекса активных ребер и (–1) в конце последовательности. Все 256 таких последовательностей образуют массив T списков треугольников (см. работу [12]).

Построение списка треугольников в каждой активной $R_{i,\,j,\,k}$-й ячейке будем выполнять в отдельном GPU-потоке (потоке “шагающих кубиков”). В данной работе создавать такие потоки предлагается с помощью программируемой тесселяции патчей-квадов. Запрограммировав графический конвейер определенным образом, можно параллельно разбивать каждый патч-квад на регулярную сетку размера до $D \times D$ вершин, где $D = L_{\max} + 1$, а $L_{\max}$ является *максимальным уровнем тесселяции* – наибольшим числом отрезков, на которое сторона патча-квада может быть разбита. В OpenGL версии 4.0 это число составляет не менее 64. В соответствии с архитектурой графического конвейера каждая получаемая в результате тесселяции вершина обрабатывается в отдельном GPU-потоке, который мы программируем для построения соответствующего ячейке списка треугольников (части моделируемой поверхности уровня). Важным преимуществом предлагаемого метода является то, что такие вершины (GPU-потоки) генерируется непосредственно внутри GPU, не тратя ресурс видеопамяти и не выходя за рамки графического конвейера, в отличие от решений, основанных на программно-аппаратной архитектуре CUDA.

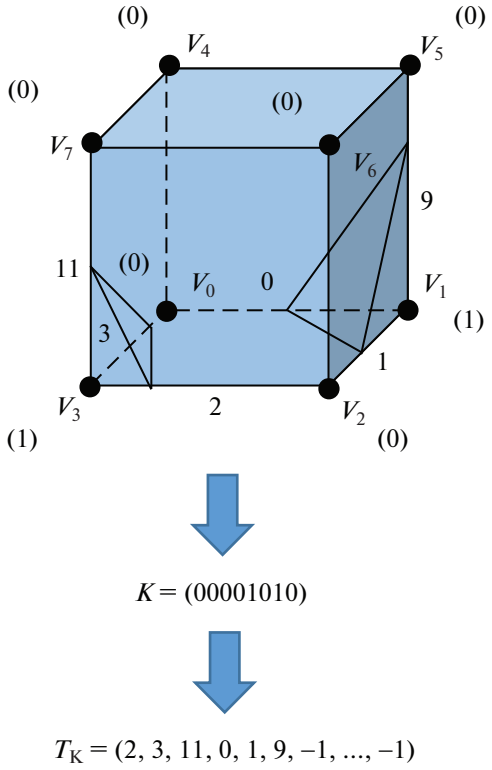

Рис. 1. Кодирование ячейки рендер-сетки.

Предлагаемая технология реализации на GPU “шагающих кубиков” включает в себя два этапа. На ***первом этапе*** (загрузка исходных данных) в видеопамять записывается вещественная 3D-текстура со значениями S вершин рендер-сетки, целочисленная 2D-текстура, в которой закодирован массив T списков треугольников, а также вершинный буфер, содержащий $m_p n_p q_p$ патчей-квадов, где $m_p = \lceil m/D \rceil$, $n_p = \lceil n/D \rceil$, $q_p = q$ (см. рис. 2). На ***втором этапе*** (визуализация) патчи-квады из вершинного буфера отправляются на графический конвейер, где распределяются между ядрами GPU и обрабатываются параллельно с помощью разработанных шейдерных программ – шейдера управления тесселяцией (TCS), шейдера вычисления тесселяции (TES), геометрического шейдера (GS) и фрагментного шейдера (FS). Далее рассмотрим их работу более подробно.

Стадия TCS-шейдера. На данной стадии выполняется вычисление двух наборов параметров.

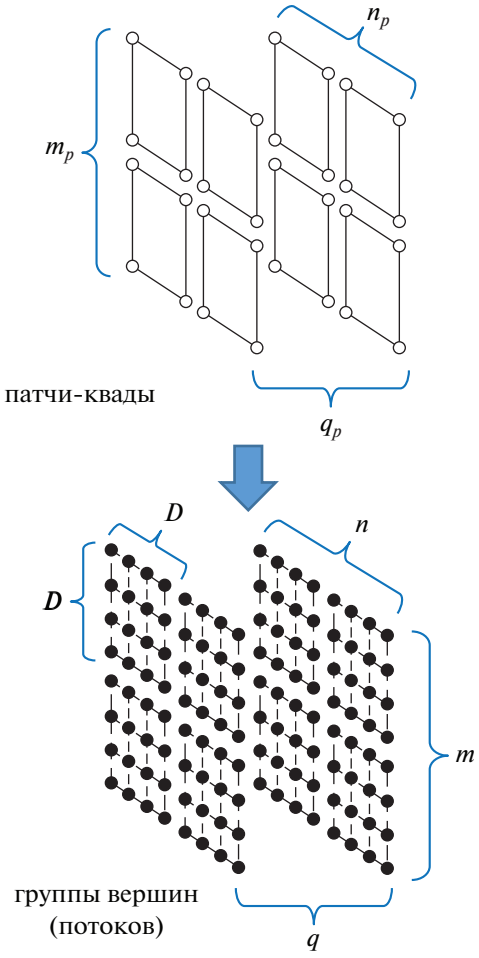

Рис. 2. Генерация потоков “шагающих кубиков”.

Первый набор является тройкой индексов (i_p, j_p, k_p) строки, столбца и слоя обрабатываемого TCS-шейдером патча в 3D массиве патчей-квадов (см. рис. 2). Эти параметры необходимы в дальнейшем для установки соответствия созданных GPU-потоков ячейкам рендер-сетки. Расчет тройки индексов (i_p, j_p, k_p) выполняется на основе встроенной переменной gl_PrimitiveID, которая является порядковым номером g патча, принимающим значения из отрезка $[0, m_p n_p q_p - 1]$:

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

где $N = m_p n_p$, а $M = Nq_p$. Второй набор включает в себя параметры, задающие ширину и высоту 2D сетки вершин (группы потоков “шагающих кубиков”). Эти ширина и высота определяются уровнями l_w и l_h тесселяции патча-квада, которые вычисляются как

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

Стадия TES-шейдера. Данный шейдер осуществляет параллельно обработку каждой вершины из 2D сетки вершин, полученной в результате тесселяции патча-квада, и устанавливает в ходе этой обработки соответствие вершины (потока) ячейке рендер-сетки. Это реализуется путем добавления обрабатываемой вершине тройки атрибутов (i_c, j_c, k_c) – индексов строки, столбца и слоя соответствующей ячейки в рендер-сетке, которые вычисляются как

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

где i и j являются индексами строки и столбца вершины в 2D сетке вершин, полученной после тесселяции патча:

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

а $(u, v) \in [0,1]$ являются нормализованными вещественными координатами обрабатываемой вершины в 2D сетке вершин (рассчитываются автоматически при тесселяции в TES-шейдере), ε – малая константа для компенсации машинной погрешности представления вещественных чисел.

Стадия GS-шейдера. Данный шейдер принимает из TES-шейдера тройку индексов (i_c, j_c, k_c) ячейки рендер-сетки и осуществляет параллельную обработку всех таких ячеек. Обработка начинается с вычисления для (i_c, j_c, k_c)-й ячейки номера конфигурации K_{i_c, j_c, k_c}:

$$K_{i_c, j_c, k_c} = \sum_{p=0}^{7} 2^p b_p,$$

где p – порядковый номер вершины в (i_c, j_c, k_c)-й ячейке (см. рис. 1), флаг b_p равен 1, если $S_p < S^*$, а в остальных случаях – 0 (S_p-значение исследуемого скалярного поля в p-й вершине ячейки). Если $K_{i_c, j_c, k_c} = 0$ или $K_{i_c, j_c, k_c} = 255$ (ячейка полностью лежит внутри или снаружи поверхности уровня), тогда GS-шейдер завершает свою работу без создания полигональной геометрии в обрабатываемой ячейке. Во всех остальных случаях: а) по номеру K_{i_c, j_c, k_c} конфигурации из текстуры T списков треугольников извлекаются списки ребер, образующих полигоны поверхности уровня внутри (i_c, j_c, k_c)-й ячейки; б) из 3D-текстуры рендер-сетки извлекаются значения S для вершин ребер и вычисляются координаты точек на этих ребрах согласно формуле (1); в) из полученных вершин выполняется построение треугольников поверхности уровня внутри (i_c, j_c, k_c)-й ячейки согласно порядку следования ребер в извлеченном списке (более подробное описание реализации эмиссии

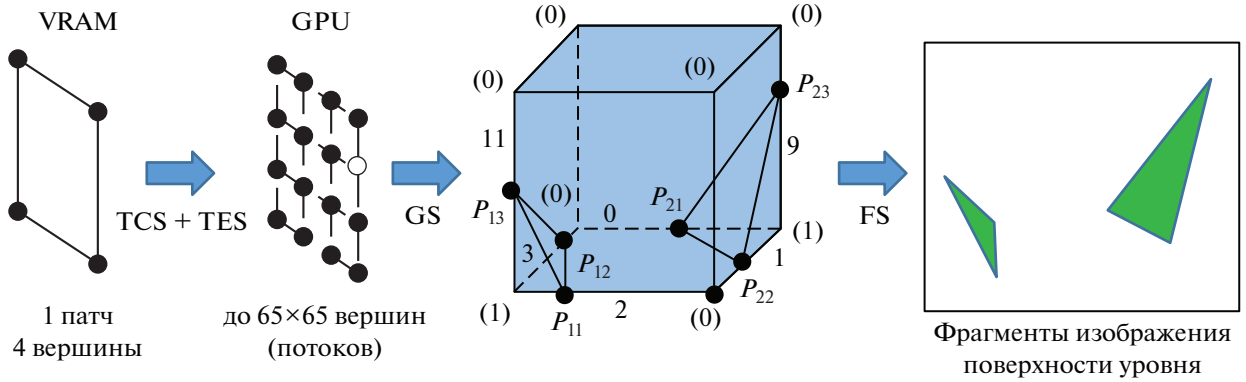

Рис. 3. Схема конвейера визуализации поверхности уровня.

треугольников в геометрическом шейдере можно найти в [29]).

Стадия FS-шейдера. Треугольники, построенные GS-шейдером проходят растеризацию (фиксированная стадия графического конвейера), в результате которой преобразуются во фрагменты изображения поверхности уровня. FS-шейдер вычисляет цвета полученных фрагментов параллельно, независимо друг от друга, на основе модели освещения Фонга [29] с направленным источником света.

На рисунке 3 показана общая схема разработанного конвейера визуализации. После прохождения всех патчей-квадов через данный конвейер в буфере кадра синтезируется результирующее изображение извлеченной поверхности уровня.

4. РЕЗУЛЬТАТЫ

На основе предложенной технологии был создан программный комплекс, реализующий извлечение полигональных моделей поверхностей уровня с помощью программируемой тесселяции. Для апробации созданного комплекса использовались трехмерные скалярные поля насыщенности воды в пористом образце нефтеносной породы, полученные в результате численного моделирования на расчетной сетке размера 100^3 ячеек. На рисунке 4 показана извлеченная поверхность, соответствующая уровню 35%-й насыщенности. Построение и визуализация полигональной модели поверхности уровня выполнялись при разрешении 3840 × 2160 с помощью видеокарты GeForce GTX 1080 Ti (3584 ядра, 11 Гб видеопамяти), средняя частота визуализации составила около 100 кадров в секунду. Полученная поверхность уровня была верифицирована путем извлечения поверхности уровня из того же массива скалярных данных с помощью системы математического и численного моделирования MATLAB [30].

Также для предложенного решения была проведена оценка накладных затрат видеопамяти (в Гб) и времени создания GPU-потоков (в миллисекундах) при увеличении размера рендер-сетки до 2000^3 ячеек. На рисунке 5 изображены графики расхода времени (T) и видеопамяти (M) для предложенной реализации (a) и для реализации распространенного подхода (b), описанного в разделе 2. По сравнению с реализацией (b) предложенное решение позволило сократить временные затраты на создание GPU-потоков в 5 раз, а накладные затраты видеопамяти – в 8 раз. На основе полученных результатов в будущем планируется провести исследование извлечения в масштабе реального времени поверхностей уровня из трехмерных скалярных полей большой протяженности (построение протяженного рельефа местности с отрицательными уклонами).

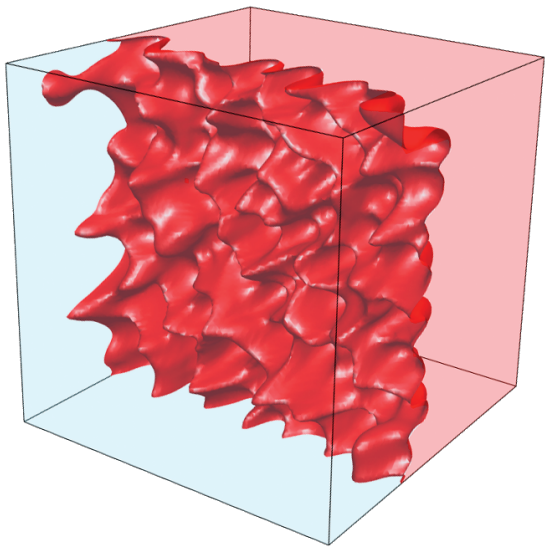

Рис. 4. Пример полученной визуализации поверхности уровня.

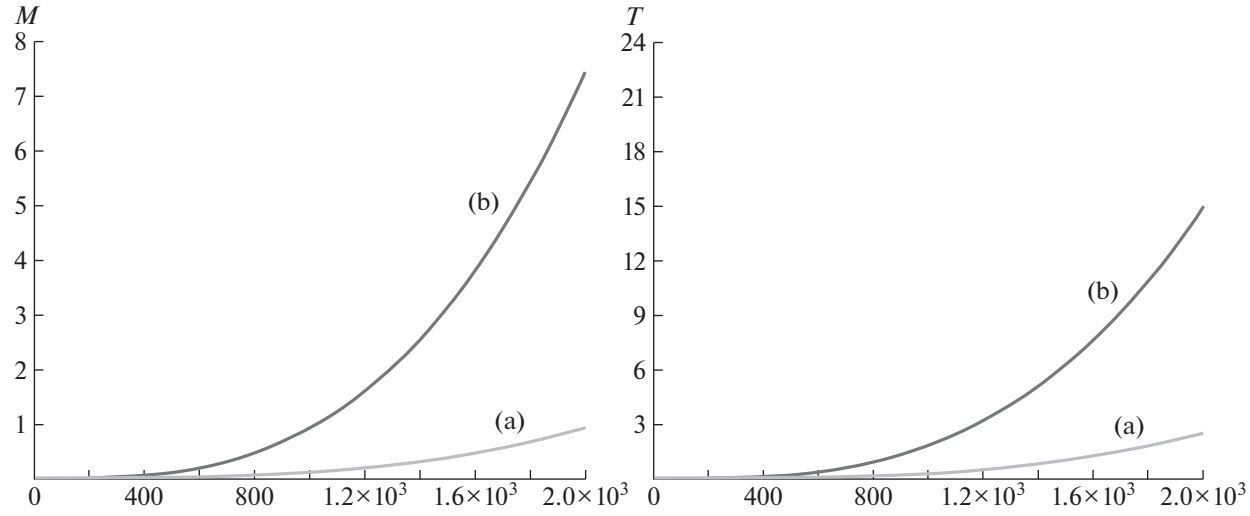

Рис. 5. Сравнение предложенного решения (a) с GPU-реализацией без тесселяционных шейдеров (b): по накладным затратам видеопамяти (слева) и по времени создания GPU-потоков (справа).

5. ЗАКЛЮЧЕНИЕ

В работе рассмотрена задача визуализации в масштабе реального времени детализированных трехмерных скалярных полей с помощью поверхностей уровня. Для преодоления выявленных ограничений, препятствующих построению в масштабе реального времени полигональных моделей поверхностей уровня сложных исследуемых объектов, предложен новый метод, основанный на выполнении на GPU-процесса извлечения поверхности уровня с помощью шейдера управления тесселяцией и шейдера вычисления тесселяции. В работе предложена эффективная технология реализации на GPU алгоритма “шагающих кубиков”, основанная на разработанном методе, которая позволяет значительно сократить временные затраты на создание GPU-потоков и накладные затраты видеопамяти по сравнению с GPU-реализацией “шагающих кубиков” без тесселяции. Созданные решения реализованы в программном комплексе, выполняющем построение и визуализацию полигональных моделей поверхностей уровня в масштабе реального времени. Разработанный комплекс был успешно апробирован на трехмерных скалярных полях насыщенности воды в образце нефтеносной породы, полученных в результате численного моделирования. Полученные изображения поверхности уровня были верифицированы с помощью системы MATLAB. В будущем планируется развитие предложенной технологии для построения и визуализации моделей рельефа местности с отрицательными уклонами.

Публикация выполнена в рамках государственного задания по проведению фундаментальных научных исследований (ГП 14) по теме (проекту) “34.9. Системы виртуального окружения: технологии, методы и алгоритмы математического моделирования и визуализации” (0065-2019-0012).

СПИСОК ЛИТЕРАТУРЫ

1. *Барладян Б.Х., Волобой А.Г., Галактионов В.А., Князь В.В., Ковернинский И.В., Солоделов Ю.А., Фролов В.А., Шапиро Л.З.* Эффективная реализация OpenGL SC для авиационных встраиваемых систем // Программирование. 2018. № 4. С. 3–10.
2. *Gavrilov N., Turlapov V.* General implementation aspects of the GPU-based volume rendering algorithm // Scientific Visualization. 2011. V. 3. № 1. P. 19–31. URL: http://sv-journal.org/2011-1/02/index.html
3. *Timokhin P., Mikhaylyuk M.* Compact GPU-based Visualization Method for High-resolution Resulting Data of Unstable Oil Displacement Simulation // Proceedings of the 29th International Conference on Computer Graphics and Vision (GraphiCon 2019), Bryansk, Russia, September 23–26. 2019. V. 2485. P. 4–6.
4. *Shakaev V.* Polygonizing volumetric terrains with sharp features // Труды 26-й Международной научной конференции GraphiCon2016, Нижний Новгород, 2016. С. 364–368.
5. *Тимохин П.Ю., Михайлюк М.В., Вожегов Е.М., Пантелей К.Д.* Технология и методы отложенного синтеза 4K-стереороликов для сложных динамических виртуальных сцен // Труды ИСП РАН. 2019. Т. 31. № 4. С. 61–72.
6. *Segal M., Akeley K.* The OpenGL Graphics System: A Specification. Version 4.6, Core Profile. The Khronos Group Inc., 2006–2018. https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf
7. *Parker S., Shirley P., Livnat Y. et al.* Interactive Ray Tracing for Isosurface Rendering // Proceedings of the IEEE Visualization 98 (VIZ’98), 1998. P. 233–238.
8. *Hadwiger M., Sigg C., Scharsach H., Bühler K., Gross M.* Real-Time Ray-Casting and Advanced Shading of Dis-

crete Isosurfaces // Computer Graphics Forum, 2005. V. 24. № 3. P. 303–312.

9. *Kim M.* GPU isosurface raycasting of FCC datasets // Graphical Models. 2013. V. 75. № 2. P. 90–101.
10. *Sanzharov V.V., Gorbonosov A.I., Frolov V.A., Voloboy A.G.* Examination of the Nvidia RTX // Proceedings of the 29th International Conference on Computer Graphics and Vision (GraphiCon 2019). 2019. V. 2485. P. 7–12.
11. *Lorensen W.E., Cline H.E.* Marching cubes: A high resolution 3D surface construction algorithm // Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques – SIGGRAPH'87, Anaheim, July 27–31. 1987. V. 21. № 4. P. 163–169.
12. *Bourke P.* Polygonising a scalar field, 1994. http://paulbourke.net/geometry/polygonise/.
13. *Kobbelt L., Botsch M., Schwanecke U., Seidel P.* Feature sensitive surface extraction from volume data // Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001. P. 57–66.
14. *Ju T., Losasso F., Schaefer S., Warren J.* Dual Contouring of Hermite Data // Proceedings of ACM Transactions on Graphics (TOG), 2002. V. 21. № 3. P. 339–346.
15. *Schmitz L., Dietrich C., Comba J.* Efficient and High Quality Contouring of Isosurfaces on Uniform Grids // Computer Graphics and Image Processing, 2009. P. 64–71.
16. *Nielson G.* Dual Marching Cubes // IEEE Visualization 2004. P. 489–496.
17. *Schaefer S., Warren J.* Dual Marching Cubes: Primal Contouring of Dual Grids // Computer Graphics Forum. 2005. V. 24. № 2. P. 195–201.
18. *Ho C.-C., Wu F.-C., Chen B.-Y., Chuang Y.-Y., Ouhyoung M.* Cubical marching squares: Adaptive Feature Preserving Surface Extraction from Volume Data // EUROGRAPHICS 2005. V. 24. № 3. P. 537–545.
19. *Manson J., Schaefer S.* Isosurfaces over simplicial partitions of multiresolution grids // Computer Graphics Forum (Proceedings of Eurographics). 2010. V. 29. № 2. P. 377–385.
20. *De Araújo B.R., Lopes D.S., Jepp P., Jorge J.A., Wyvill B.* A Survey on Implicit Surface Polygonization, 2015, ACM Computing Surveys. V. 47. № 4. P. 1–39.
21. *Matsumura M., Anjo K.* Accelerated Isosurface Polygonization for Dynamic Volume Data Using Programmable Graphics Hardware // Proceedings of SPIE-IS&T Electronic Imaging, Visualization and Data Analysis. 2003. V. 009. P. 145–152.
22. *Hansen C., Johnson C.* (editors). Visualization Handbook. Elsevier Press, 2004. 984 p.
23. *Goetz F., Junklewitz T., Domik G.* Real-Time Marching Cubes on the Vertex Shader // Proceedings of Eurographics 2005. P. 1–4.
24. *Tatarchuk N., Shopf J., DeCoro C.* Real-Time Isosurface Extraction Using the GPU Programmable Geometry Pipeline // Proceedings of ACM SIGGRAPH 2007 Courses. 2007. P. 122–137.
25. *Dyken C., Ziegler G., Theobalt C., Seidel H.-P.* High-speed Marching Cubes using HistoPyramids // Computer Graphics Forum. 2008. V. 27. № 8. P. 2028–2039.
26. *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 // Proceedings of the IASTED International Conference on Automation, Control, and Information Technology (ACIT 2010), 2010. P. 236–240.
27. *Chen J., Jin X., Deng Z.* GPU-based polygonization and optimization for implicit surfaces // The Visual Computer. 2015. V. 31 (2). P. 119–130.
28. *Михайлюк М.В., Тимохин П.Ю., Мальцев А.В.* Метод тесселяции на GPU рельефа Земли для космических видеотренажеров // Программирование, 2017. № 4. С. 39–47.
29. *Bailey M., Cunningham S.* Graphics Shaders: Theory and Practice, Second Edition. CRC Press, 2011. 518 p.
30. MATLAB Documentation. Volume Visualization. https://www.mathworks.com/help/matlab/ref/isosurface.html