ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2015. Т. 19, № 4 (70). С. 84–94 http://journal.ugatu.ac.ru

УДК 004.92

ПРИМЕНЕНИЕ БИБЛИОТЕКИ OPENGL ДЛЯ ВИЗУАЛИЗАЦИИ РЕЗУЛЬТАТОВ ЧИСЛЕННОГО МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ НА СЕТКАХ БОЛЬШОЙ РАЗМЕРНОСТИ

М. Р. БАДРЕТДИНОВ[1], Т. Р. БАДРЕТДИНОВ[2], О. С. БОРЩУК[3]

[1] BadretdinovMR@ufanipi.ru, [2] BadretdinovTR@ufanipi.ru, [3] BorschukOS@ufanipi.ru

[1,2] ФГБОУ ВПО «Уфимский государственный авиационный технический университет» (УГАТУ)
[3] ООО «РН-УфаНИПИнефть»

Поступила в редакцию 4.11.2015

Аннотация. В работе предложены новые подходы к визуализации сеточных моделей больших размеров (более 100 млн. ячеек) с использованием языка шейдеров библиотеки OpenGL. Предложены три подхода к визуализации данных на структурированных сетках, отличающиеся степенью использования графических процессов для проведения вычислений, дан их сравнительный анализ. На основе предложенных методов разработан и представлен прототип визуализатора.

Ключевые слова: компьютерная графика; OpenGL, визуализация сеточной модели, геометрические шейдеры, imposter объекты.

ВВЕДЕНИЕ

Язык шейдеров OpenGL (OpenGL Shading Language, GLSL) в настоящее время стал фундаментальной основой и неотъемлемой частью программирования с использованием библиотеки OpenGL. Его применение раскрывает новые возможности для превращения прежде фиксированного конвейера обработки графики в программный (т. е. на каждом шаге конвейера выполняется подпрограмма – шейдер). Благодаря GLSL можно более эффективно использовать возможности графического процессора (Graphics Processor Unit, GPU) для реализации отображения больших объемов данных и даже для проведения необходимых вычислений [1].

В данной статье предлагается несколько подходов для решения проблемы визуализации больших ($\approx 10^6$ ячеек) и гигантских ($> 10^8$ ячеек) сеточных моделей на типовых персональных компьютерах (ПК), оснащенных дискретной графической картой, представлен сравнительный анализ этих подходов. Предлагаемые подходы к визуализации сеточной модели оптимизируют объем используемой графической памяти и скорость «отрисовки» и основаны на применении геометрического шейдера [2]. Также в статье представлен алгоритм визуализации 3D трубок высокого качества без увеличения нагрузки на графический процессор.

Для проведения тестирования нами был реализован прототип модуля визуализации результата численного моделирования трехфазной фильтрации флюидов в поровой среде (геолого-гидродинамическое моделирование нефтяных месторождений, ГГДМ) (рис. 1).

Отметим, что все представленные алгоритмы могут быть использованы не только для визуализации ГГДМ, но и для любых других результатов численного математического моделирования на структурированных 3D сетках. Алгоритмы визуализации скважин также могут быть использованы для представления поля скоростей в виде трубок тока.

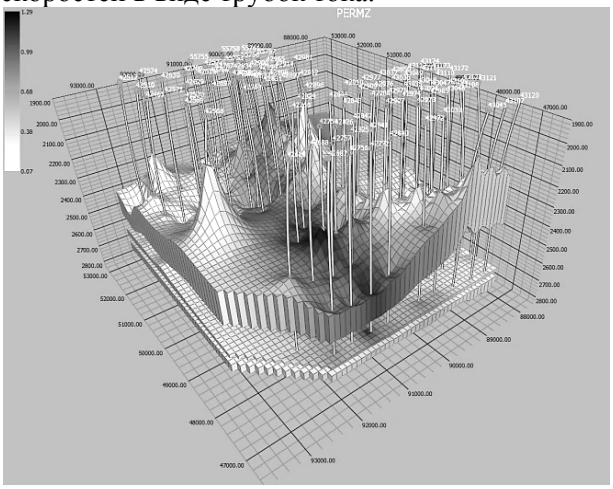

Рис. 1. Прототип модуля визуализации ГГДМ. Пример работы визуализатора

ПОСТАНОВКА ЗАДАЧИ

Целью работы являлась разработка оптимизированных алгоритмов визуализации результатов численного математического моделирования на структурированных 3D сетках, использующих возможности языка шейдеров (GLSL) библиотеки OpenGL.

При моделировании ГГДМ в основном используются сетки двух видов: угловая геометрия GRDECL [3] и 2,5D ячейки Вороного (PEBI) [4]. Дискретизация системы уравнений сохранения массы в частных производных [5] проводится с использованием метода конечного объема [6], это означает, что сетка является блочно-центрированной (значения неизвестных ищутся не в узлах сетки, а в центрах блоков).

Для отображения результатов моделирования необходимо эффективно выполнять следующие базовые операции:

- визуализация ячеек расчетной сетки с раскраской в соответствии со значением некоторого скалярного поля ($> 10^8$ ячеек),
- быстрая смена значения скалярного поля,
- визуализация геометрии сеточной модели (ребер ячеек),
- визуализация срезов по ячейкам и сечениям,
- фильтрация ячеек, удовлетворяющих условию на скалярное поле (например, отображать только те ячейки, где давление больше 100 атм.),
- визуализация траекторий скважин или трубок тока,
- отображение разломов, выклиниваний и других геологических особенностей строения месторождений.

ВОЗМОЖНОСТИ И ОГРАНИЧЕНИЯ БИБЛИОТЕКИ OPENGL

Библиотека OpenGL позволяет эффективно визуализировать только графические примитивы (точки, отрезки, треугольники), поэтому любой визуализируемый объект должен быть представлен в виде набора этих примитивов. Кроме того, в OpenGL параметры визуализации (цвет, вектор нормали) возможно задать только для вершин примитивов, а не для самих примитивов, что необходимо при использовании блочно-центрированной геометрии. Часто для уменьшения занимаемой графической памяти используют индексный подход, в котором отдельно задаются координаты и параметры вершин, а задание примитивов происходит посредством задания индексов вершин. Например, рассмотрим куб со стороной 1. Пусть его вершины упорядочены следующим образом: $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, $(1,1,0)$, $(0,0,1)$, $(1,0,1)$, $(0,1,1)$, $(1,1,1)$ (т.е. вершина (0,0,0) имеет номер 1, и т.д.). Куб имеет 6 квадратных граней, и, значит, для их визуализации необходимо использовать 12 примитивов (треугольников): $(1,2,3)$, $(2,4,3)$, $(2,8,6)$, $(2,4,8)$, $(1,5,6)$, $(1,6,2)$, $(1,3,7)$, $(1,7,5)$, $(3,4,8)$, $(3,8,7)$, $(5,7,8)$, $(5,8,6)$. Каждый треугольник задан тремя номерами вершин куба. Аналогично для визуализации ребер необходимо задать 12 отрезков: $(1,2)$, $(1,3)$, $(2,4)$, $(3,4)$, $(5,6)$, $(5,7)$, $(6,8)$, $(7,8)$, $(1,5)$, $(2,6)$, $(3,7)$, $(4,8)$.

Начиная с версии OpenGL 3.3, в состав динамического конвейера обработки графических данных включен геометрический шейдер, и поэтому конвейер состоит из следующих стадий:

- вершинный шейдер – обработка каждой вершины в отдельности без доступа к данным других вершин, обычно используется для преобразования координат, вектора нормали, цвета;
- геометрический шейдер – обработка данных по примитиву в целом, есть доступ ко всем вершинам примитива, возможна генерация новых примитивов.
- пиксельный шейдер – вызывается для каждого пикселя примитива в отдельности, имеет доступ только к интерполированным внутри примитива параметрам, используется для расчета освещения и других эффектов.

Добавление геометрического шейдера позволило существенно повысить возможности по обработки геометрии на GPU. Однако в текущем поколении графических карт реализация геометрического шейдера недостаточно эффективна и имеет существенные ограничения на количество генерируемых примитивов и параметров. Основные производители графических карт NVIDIA и AMD заявляют, что производительность геометрического шейдера будет значительно повышена в следующем поколении GPU.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОДХОДОВ К ВИЗУАЛИЗАЦИИ

Классическим подходом к визуализации расчетной сетки является расчет на центральном процессоре (Central Processor Unit, CPU) координат вершин сетки и полигональной модели поверхности (набора треугольных примитивов, образующих замкнутую внешнюю оболочку расчетной сетки)[7]. Если требуется отображение ребер сетки, то дополнительно произ-

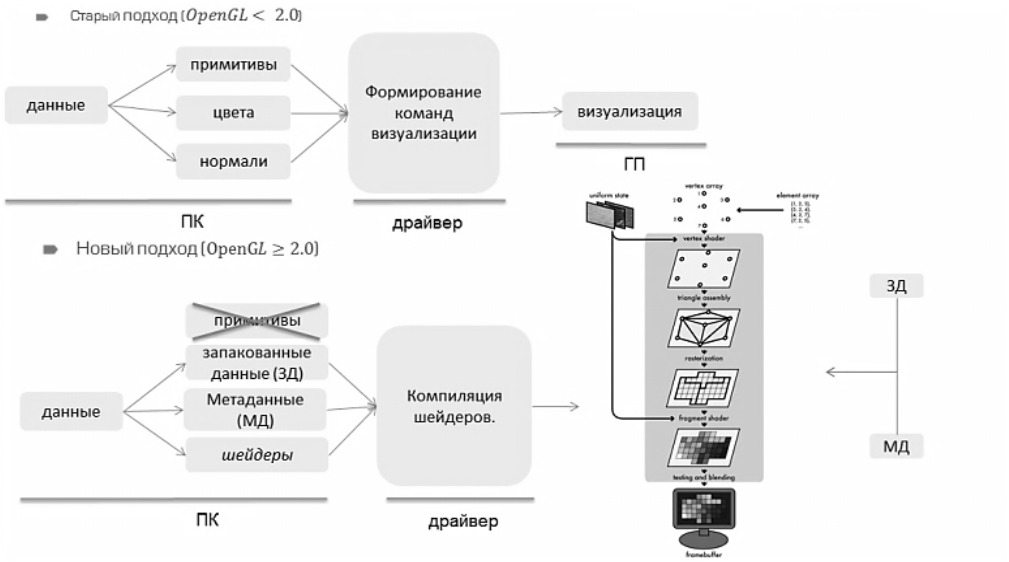

Рис. 2. Упрощенная схема визуализации. Сравнение старого и нового подхода к визуализации OpenGL

водится расчет координат ребер сетки. В настоящее время GPU имеет преимущество по производительности относительно CPU, и естественным является желание часть работ по расчету координат вершин и индексов примитивов (треугольников) передать на GPU. Использование динамического конвейера и шейдеров OpenGL предоставляет такие возможности. Ранее используемый подход к визуализации обладал рядом существенных недостатков, связанных с аппаратными и программными ограничениями. Недостаток памяти на графических процессорах (GPU) и фиксированный цикл графических программных интерфейсов не давал необходимой свободы для работы с визуализируемыми данными, приходилось работать с массивами вершин (рис. 2). Использование шейдеров ввело в данный процесс дополнительный блок, различные реализации которого обсуждаются далее.

ИСХОДНЫЕ ДАННЫЕ

Для описания расчетной сетки будем использовать формат GRDECL [3]. Формат GRDECL позволяет описывать структурированную 3D сетку из N_x, N_y, N_z ячеек по осям Ox, Oy, Oz соответственно. Ячейки сетки представляют собой шестигранники (искаженные параллелепипеды, возможно с вырожденными гранями). Для описания структуры сетки задаются $(N_x + 1) \times (N_y + 1)$ субвертикальных векторов (рис. 4), они являются направляющими, на которых расположены вершины ячеек. В дополнение к направляющим задаются координаты Z восьми вершин для каждой ячейки (рис. 3). Координаты вершин ячейки вычисляются из пересечения соответствующих направляющих и уровней Z. Это позволяет задавать сетки, стыкованные в направлении Ox и Oy и не стыкованные в направлении Oz.

Формат GRDECL часто используется при геолого-гидродинамическом моделировании, так как позволяет учесть наличие разломов, выклиниваний и других особенностей геологического строения месторождений. На практике описание сетки производится тремя массивами:

- массив COORD имеет размер $6 \times \times (N_x + 1) \times (N_y + 1)$, содержит $(N_x + 1) \times \times \left(N_y + 1\right)$ координат (X_1, Y_1, Z_1, X_2, Y_2, Z_2) точек, определяющих субвертикальные прямые – направляющие;
- массив ZCORN размера $8 \times N_x \times N_y \times \times N_z$ – содержит координаты Z восьми вершин ячеек;
- массив ACTNUM размера $N_x \times N_y \times \times N_z$ – это массив «маска», определяет активность ячеек, что означает, используется ли она для расчета и необходимо ли ее отображение при визуализации.

Преимущество формата представления GRDECL заключается в уменьшении объема используемой оперативной памяти по сравнению со стандартными способами хранения. Например, нестыкованная сетка размерности $N_x = 100$, $N_y = 100$, $N_z = 100$ в формате GRDECL будет занимать приблизительно 36 МБ, а в стандартном формате – 100 МБ.

Для визуализации расчетной сетки необходимо превратить ее в набор примитивов (треугольников), при этом координаты вершин должны быть вычислены с использованием данных из массивов ZCORN и COORD. Разработанные подходы 2 и 3 переносят вычисления

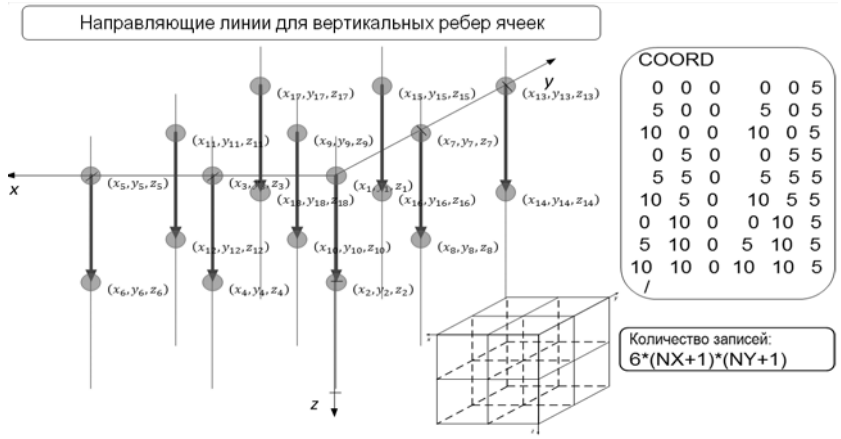

Рис. 3. Формат GRDECL. Направляющие вектора для субвертикальных ребер ячеек (COORD)

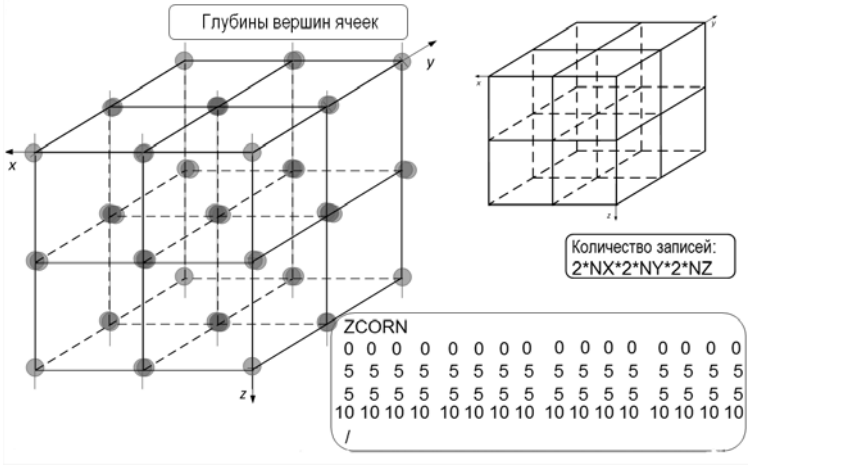

Рис. 4. Формат GRDECL. Глубины вершин ячеек (ZCORN)

координат вершин треугольников на видеокарту, благодаря чему уменьшаются требования к количеству графической оперативной памяти и вычислительным ресурсам CPU. Это, в конечном итоге, и позволяет визуализировать ранее не доступные объемы данных.

ВИЗУАЛИЗАЦИЯ РАСЧЕТНОЙ СЕТКИ: ПЕРВЫЙ ПОДХОД

Данный подход применим, если сетка является стыкованной либо когда наличие разломов и выклиниваний не является критически важным для визуального анализа данных.

Для стыкованной сетки формат COORD, ZCORN является избыточным, так как для хранения координат достаточно массива вершин размером $3 \times (N_x + 1) \times (N_y + 1) \times (N_z + +1)$, содержащего координаты (X, Y, Z) для каждой из $(N_x + 1) \times (N_y + 1) \times (N_z + 1)$ вершин. Построив данный массив на CPU, остается определить видимые грани всех ячеек сетки (грань считается видимой, если она находится на границе расчетной сетки или для двух ячеек, которые она разделяет, выполнено условие, что одна является активной, а другая – нет).

В связи с ограничениями библиотеки OpenGL цвет примитива определяется либо цветом «первой» вершины, либо интерполяцией по трем вершинам. Однако окраска ячейки в один цвет возможна задаем цвета только двух (из восьми) ее вершин. Для этого была разработана схема отрисовки ячеек сетки в два этапа (рис. 5). Для каждой вершины сетки с индексом $(i,\ j,\ k)$ передаются два цвета для ячейки $(i,\ j,\ k)$ и для ячейки $(i-1, j-1, k-1)$. Треугольники для граней $-X$, $-Y$, $-Z$ (6 треугольников, первый этап отрисовки) задаются таким образом, что для ячейки с индексом $(i,\ j,\ k)$ первой вершиной является вершина сетки с индексом $(i,\ j,\ k)$, а для граней $+X$, $+Y$, $+Z$ (6 треугольников, второй этап отрисовки) вершина $(i+1, j++1, k+1)$.

Для экономии оперативной памяти используется сжатие данных для вершин в два 4-байтных беззнаковых целых числа (Таблица 1). Задается палитра (256 цветов по 4 байта) для цвета и используются индексы (1 байт) для каждой вершины. Распаковка данных ведется в вершинном шейдере, т. е. на графическом процессоре (GPU), где идет их дальнейшая обработка.

Таблица 1

Формат упаковки данных для каждой вершины (8 байт)

2B	2B	2B	2B	
X	Y	Z	Цвет1	Цвет2

Цвет1, Цвет2 – индекс цвета для первой и второй итерации рисования, выбирается в шейдере. Палитра передается как 256 RGB цветов по 3 байта, т. е. 768 байт на палитру.

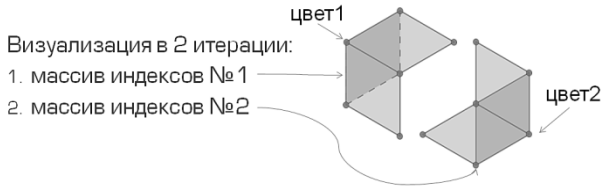

Рис. 5. Разделение ячейки на примитивы и отрисовка в два этапа

Если оценить расход памяти для оболочки модели (все ячейки активны) размерности (1000, 1000, 100), т.е. 10^8 ячеек, то получим:

1. $\frac{1001\times1001\times101\times8}{1024^2} \approx 772$ Мб – на координаты вершин и индексы цвета;
2. $\frac{2\times1000\times1000+4\times1000\times100}{1024^2} \times 6 \times 4 \approx 55$ Мб – на индексы;
3. $256 \times 3 = 768$ байт $\approx 7e-4$ Мб – на палитру.

Всего 827 Мб.

Таким образом, буфер вершин для 10^8 ячеек стыкованной модели (оболочка) занимает приблизительно 827 Мб. Отметим, что если не сжимать данные, то объем необходимой памяти составит 1,93 Гб. Таким образом, выигрыш относительно классического метода составляет $7-8$ раз.

ВИЗУАЛИЗАЦИЯ РАСЧЕТНОЙ СЕТКИ: ВТОРОЙ ПОДХОД

Второй подход учитывает наличие нестыкованных ячеек (отображает разломы).

Идея подхода состоит в следующем: проводить расчет координат вершин из массивов COORD, ZCORN на GPU, создавать примитивы (треугольники) прямо на GPU за счет использования геометрического шейдера.

Массивы COORD, ZCORN передаются непосредственно на видеокарту. Для каждой ячейки расчетной сетки определяется видимость ее граней, грань считается видимой, когда выполнено любое из трех условий:

- она находится на границе расчетной сетки;
- для двух ячеек, которые она разделяет, выполнено условие, что одна является активной, а другая нет;
- грань соединяет нестыкованные ячейки (т.е. не совпадает хотя бы одна из координат Z грани для ячейки справа и слева).

Так как мы используем геометрический шейдер для генерации примитивов, то в качестве входных данных будем использовать номера ячеек $(i + j \times N_x + k \times N_x \times N_y)$ с дополнительной информацией о видимости граней и цвете ячеек.

В буфер вершин заносится информация по всем видимым ячейкам (хотя бы одна грань видима) (табл. 2.).

Таблица 2

Запаковка данных для 1 ячейки (8 байт)

4B	2B	1B	1B
Номер ячейки	Зарезервировано	Битовая маска видимости граней (-X, X, -Y, Y, -Z, Z)	Цвет

Также на видеокарту передается палитра, определяемая как и в первом подходе, и размерность модели по осям, что позволяет определить индекс (i, j, k) из номера ячейки. Запакованные данные распаковываются в геометрическом шейдере на видеокарте. По индексу ячейки, размерности модели и COORD, ZCORN получаются координаты вершин для видимых граней ячейки. В геометрическом шейдере формируются треугольники для отрисовки на основе распакованных вершин. Если оценить расход памяти для оболочки модели (все ячейки активны) размерности (1000, 1000, 100), т.е. состоящей из 10^8 ячеек, то получим:

1. $6 \times (\mathrm{nx}+1) \times (\mathrm{ny}+1) \times$ 4 байта ≈ 23 Мб – на COORD;
2. $8 \times \mathrm{nx} \times \mathrm{ny} \times \mathrm{nz} \times$ 4 байта $\approx 2{,}98$ Гб – на ZCORN;
3. Приблизительно 9 Мб – на индексы визуализируемого куба;
4. $256 \times 3 = 768$ байт – на палитру.

Всего ≈ 3 Гб.

Как можно заметить, больше всего памяти требует массив Z координат вершин ячеек ZCORN. Его можно сжать, запаковав каждые 2 значения в 4 байта (Таблица 3), чтобы он занимал меньше места, а затем распаковать в шейдере. Таким образом, сжатый ZCORN занимает 1,49 Гб, что в итоге дает нам 1,5Гб для всей ГГДМ.

Таблица 3

Сжатый массив ZCORN

4B		4B		4B		4B	
Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8

ВИЗУАЛИЗАЦИЯ РАСЧЕТНОЙ СЕТКИ: ТРЕТИЙ ПОДХОД

При использовании второго подхода возникли проблемы с производительностью GPU, связанные с большой нагрузкой на геометрический шейдер. В современных видеокартах функциональность геометрического шейдера реализована недостаточно эффективно, когда геометрический шейдер генерирует большое или нестабильное количество примитивов. Обе данные проблемы актуальны для второго подхода. Для оптимизации скорости работы был разработан третий подход, основная идея которого заключается в снижении нагрузки на геометрический шейдер за счет переноса части работ по генерации примитивов на CPU.

В третьем подходе массивы COORD, ZCORN передаются на видеокарту аналогично второму подходу. Вместо индексов вершин (второй подход) на видеокарту передаются индексы массива Z координат ячеек ZCORN для треугольников (т. е. определение видимости граней и деление их на примитивы производится на CPU). На основе одномерного индекса массива ZCORN и его размерности в вершинном шейдере вычисляется индекс визуализируемой ячейки $(i,\ j,\ k)$, на основе этой информации в геометрическом шейдере строятся треугольники для отрисовки (рис. 6).

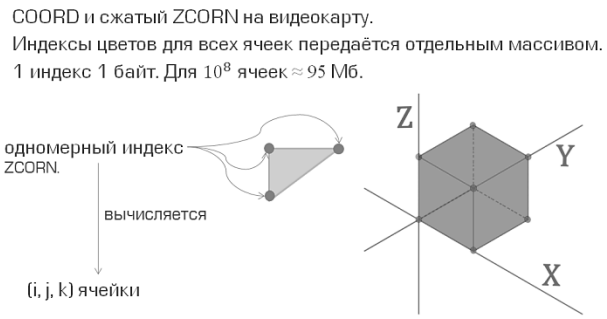

Рис. 6. Построение треугольников по вершине ячейки

Если оценить расход памяти для оболочки модели (все ячейки активны) размерности (1000, 1000, 100), состоящей из 10^8 ячеек, то получим:

1. $6 \times (nx + 1) \times (ny + 1) \times 4$ байта $\approx$ 23 Мб – на COORD;
2. $\frac{1000 \times 1000 \times 100 \times 4}{1024^3} \times 4 \approx 1{,}49$ Гб – на ZCORN;
3. $256 \times 3 = 256 \times 3 = 768$ байт – на палитру;
4. $\frac{1000 \times \mathbf{1000} \times \mathbf{100}}{1024^2}$ – на индексы цвета кубов.

Всего $\approx 1{,}66$ Гб.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОДХОДОВ ВИЗУАЛИЗАЦИИ

Для сравнительного анализа (табл. 4) использовался ноутбук Acer (Core i5 480M 2660МГц, 4Гб DDR3, GeForce GT 540M). Сравнение использованной видеокарты с новейшими приведено в табл. 4. Для тестов была использована нестыкованная расчетная сетка с размерностью 8 614 584 ячеек (198 x 292 x 149). Лучший результат по занимаемой памяти (наименьший объем) получился у первого подхода (рис. 7). Это было ожидаемо, т. к. в первом подходе мы не учитываем наличие не стыкованных ячеек (возможные разломы). Второй метод визуализируется дольше остальных. Это связано с большой нагрузкой на геометрический шейдер – предыдущее поколение графических процессоров не справляется с нагрузкой. Можно заметить, что современные графические процессоры превосходят в 20–30 раз по количеству вычислительных ядер тот, на котором проводилось тестирование (табл. 4).

Таблица 4

Сравнение современных графических процессоров

	GeForce GTX TITAN X	GeForce GTX 980	GeForce GT 540M
Кол-во шейдерных ядер	3072	2048	96
Объем памяти	12 GB	4 GB	1 GB
Интерфейс памяти	384-bit GDDR5	256-bit GDDR5	128-bit GDDR3
Макс. Пол. проп. памяти	336,5 GB/s	224 GB/s	28.8 GB/sec
Цена	$ 1000	$ 550	-

ВИЗУАЛИЗАЦИЯ РЕБЕР ЯЧЕЕК СЕТКИ

Для анализа данных часто необходимо видеть структуру (геометрию) расчетной сетки, этого можно добиться визуализацией ребер ячеек расчетной сетки. Однако традиционные методы не подходят для визуализации расчетной сетки больших ГГДМ в силу того, что они либо неэффективны по памяти, либо неэффективны по скорости. Первое связано с дополнительными расходами на текстурные координаты на каждую

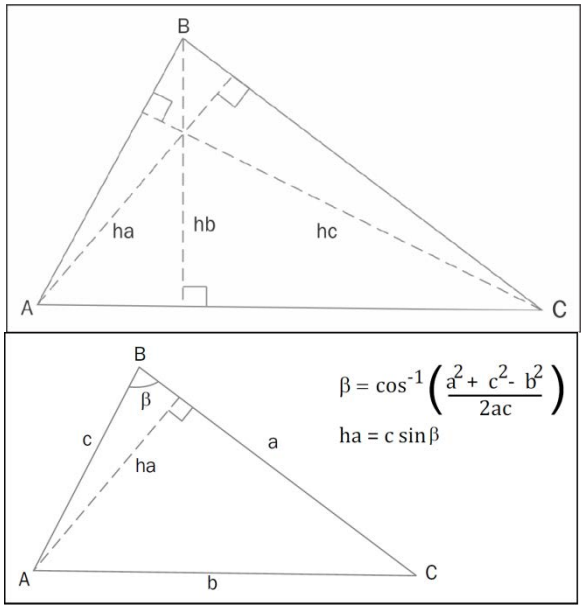

Рис. 8. Вычисление расстояния до границ полигона

вершину. Второе связано с необходимостью рисовать модель в два цикла: сначала полигональную модель, а затем ребра ячеек. Кроме того, в случае второго подхода при масштабировании может наблюдаться муар (т. е. визуальное искажение структуры) и сокрытие полезной информации. Последнее обусловлено тем, что толщина линий в OpenGL не зависит от расстояния до наблюдателя, вследствие чего для удаленных/маленьких ячеек цвет, обозначающий значение скалярного поля, будет перекрываться цветом линий на границах ячейки, что будет приводить к потере полезной информации.

Шейдеры позволяют визуализировать полигональную модель и ребра ячеек в одном цикле визуализации без дополнительной ощутимой нагрузки по памяти и скорости визуализации [8, 9]. Кроме того, в шейдерах возможно определить зависимость толщины линии от размера ячейки на экране, таким образом решив проблему потери информации о скалярном поле в удаленных ячейках (рис. 9).

Особенность нашей реализации заключается в том, что нам надо рисовать не каждое ребро примитива (треугольника), а только внешние ребра грани ячейки (2 треугольника с общей границей). Грань ячейки – четырехугольник, который для визуализации разбивается на два треугольника. Вершины треугольников должны быть пронумерованы таким образом, чтобы последнее ребро проходило по центру грани ячейки. Т. е. для каждой грани ABCD должны задаваться треугольники ABC и DCA. Это необходимо для того, чтобы не рисовать диагональ у граней ячейки. Тогда для каждого треугольника мы можем определить одни и те же инструкции для рисования граней в шейдерах (рис. 7):

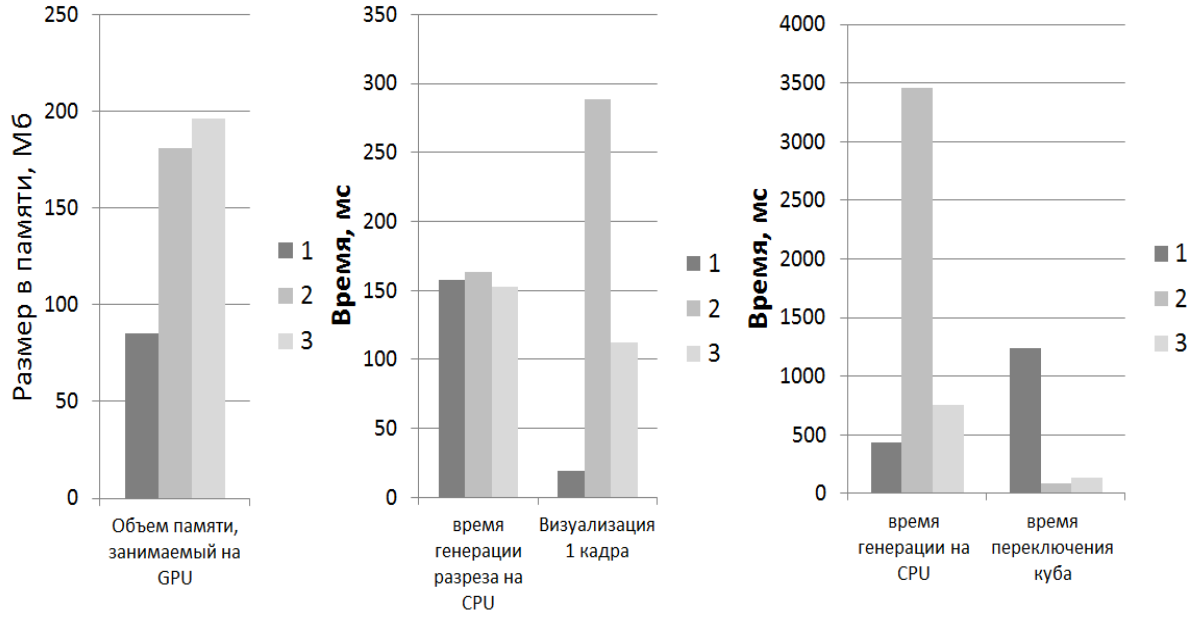

Рис. 7. Сравнение работы алгоритмов для нестыкованной модели: 8 614 584 ячеек (198x292x149)

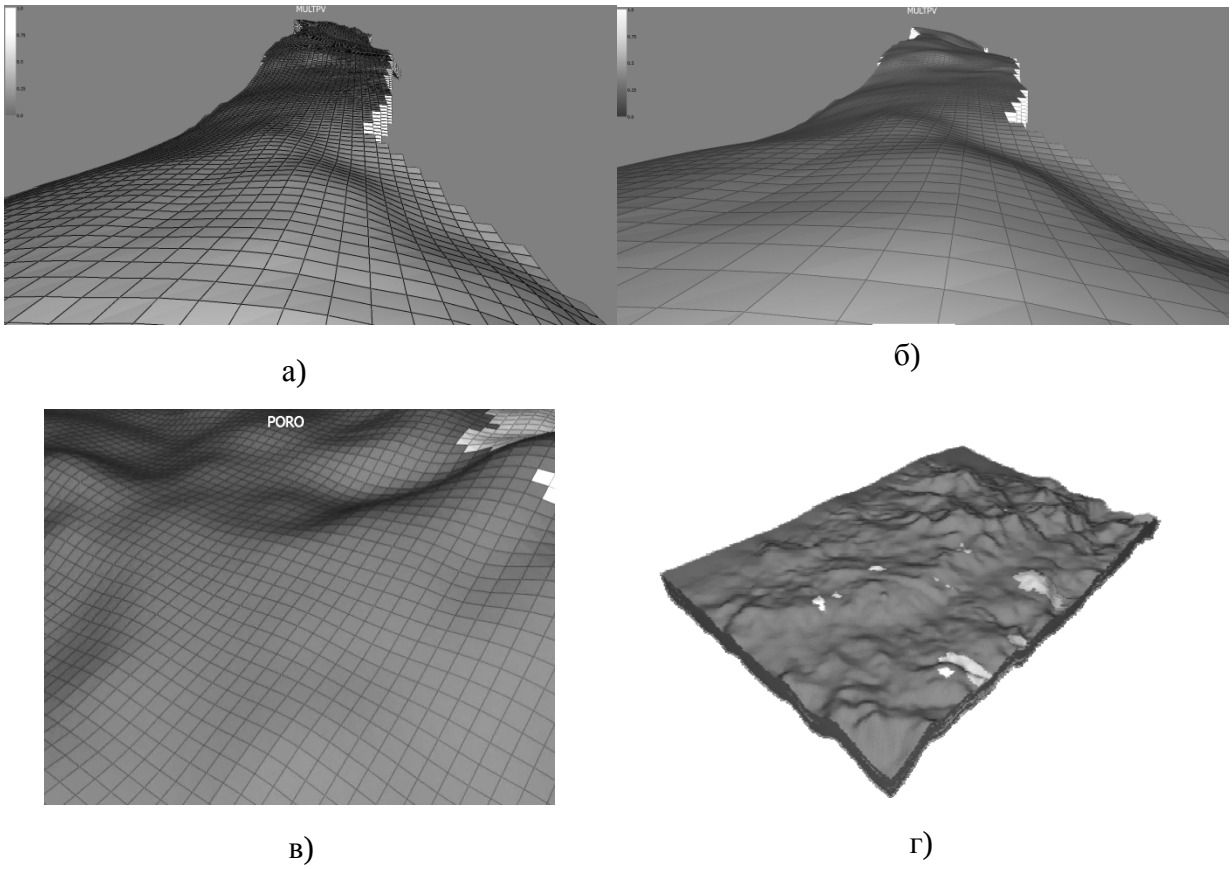

Рис. 9. Примеры визуализации сеточной модели ГГДМ: а) без угасания, б) с угасанием, в) с угасанием при приближении, г) с угасанием при удалении

1. Для каждого треугольника ABC рассчитываются высоты ha, hb, hc (рис. 8).
2. Вычисляется площадь треугольника S_{ABC} в экранных координатах, для того чтобы не рисовать грани у маленьких треугольников.
3. Каждой вершине присваивается вектор d, значения в котором показывают расстояние от данной вершины до каждой грани (эти значения будут интерполироваться между геометрическим и фрагментарным шейдером):
$$A: d = vec3(ha, 0,0),$$
$$B: d = vec3(0,0, hc),$$
$$C: d = vec3(0, hb, 0);$$
4. Во фрагментарном шейдере фрагменты, для которых площадь треугольника на экране мала $S_{ABC} < eps$, закрашиваются цветом ячейки.
5. Для остальных фрагментов определяется ближайшая грань, если это грань AC (второе значение в векторе d), тогда фрагмент закрашивается цветом ячейки, иначе, если ближайшая грань AB или BC, используется механизм смешивания цвета границы с цветом ячейки в зависимости от удаления фрагмента от границы, что равносильно уменьшению толщины линии. Существуют различные алгоритмы смешивания [8, 9], мы использовали функцию стандарта OPENGL – smoothtep.

Представленный алгоритм обладает тремя преимуществами относительно классического: отрисовка происходит в один проход без существенного влияния на производительность; отсутствуют паразитные эффекты муара и скрытия цветовой информации; линии получаются сглаженными (Anti-aliasing), что положительно сказывается на общем качестве картинки.

ВИЗУАЛИЗАЦИЯ 3D ТРУБОК

Как упоминалось ранее, библиотека OpenGL может работать только с графическими примитивами (точки, отрезки, треугольники). Для отображения в трехмерном пространстве таких объектов, как шар (используются для визуализации маркеров) или трубка (визуализация траекторий скважин и трубок тока) необходимо большое количество полигонов. Чем больше полигонов необходимо отрисовать, тем больше нагрузка на графический процессор.

Шейдеры OpenGL позволяют писать алгоритмы освещения таким образом, чтобы двухмерный объект выглядел как трехмерный. Так для визуализации шара достаточно всего лишь

нарисовать 2 треугольника [13]. Для этого в геометрическом шейдере треугольники необходимо поворачивать таким образом, чтобы их нормаль была направлена на камеру. Зная радиус во фрагментарном шейдере мы можем отсекать фрагменты, которые не попадают в шар, а остальные фрагменты закрашивать в соответствии с расстоянием до камеры. Такие возможности часто используют при визуализации молекулярных моделей [10, 11, 12].

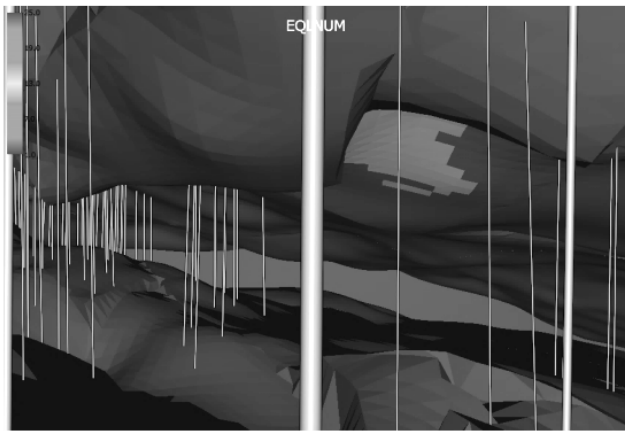

Рис. 9. Визуализация скважин, маркеров и текста

Нами был разработан алгоритм для визуализации 3D трубок, который в качестве входных параметров берет опорные точки траектории трубки, на их основе создаёт плоскости, стыкованные друг с другом, и за счет освещения придает этим плоскостям объем. В результате получается траектория идеальной цилиндрической формы (рис. 10).

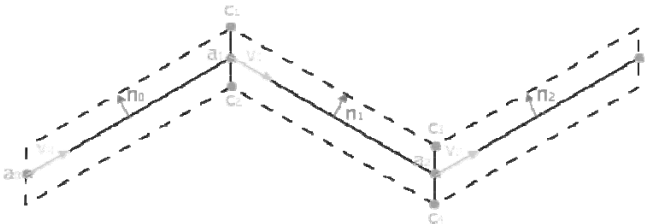

Рис. 10. Генерация точек цилиндра на основе 4 входных точек траектории

Визуализация осуществляется по сегментам (рис. 11). Для создания одного сегмента $(C_1C_2C_3C_4)$ требуется четыре опорных точки траектории $(a_0a_1a_2a_3)$ и радиус R цилиндра. Сегмент строится между точками a_1 и a_2, в то время как a_0 и a_3 являются вспомогательными точками для определения границ сегмента. Следующий сегмент строится по точкам $(a_1a_2a_3a_4)$. Аналогичным образом, со смещением на одну точку, берутся точки для следующих сегментов. Для рисования 3D трубок мы используем режим визуализации OpenGL, который позволяет добиться подобного поведения - GL_LINE_STRIP_ADJACENCY.

Следующие вычисления производятся для генерации плоскости сегмента в геометрическом шейдере:

1. Опорные точки траектории переводятся из мировой системы координат в систему координат наблюдателя:
   $a_i = ModelViewMatrix * a_i \; for \; i \in [0:3]$;
2. Определяется направление сегмента (понадобится для определения цвета):
   $vec3 \; way = normalize(a_1.xyz - a_2.xyz)$;
3. Вычисляются направляющие вектора:
   $vec2 \; v_0 = normalize(a_0.xy - a_1.xy)$,
   $vec2 \; v_1 = normalize(a_1.xy - a_2.xy)$,
   $vec2 \; v_2 = normalize(a_2.xy - a_3.xy)$;
4. Вычисляются нормали к отрезкам между опорными точками траектории:
   $vec2 \; n_0 = vec2 \; (v_0.y, -v_0.x)$,
   $vec2 \; n_1 = vec2 \; (v_1.y, -v_2.x)$,
   $vec2 \; n_2 = vec2 \; (v_1.y, -v_2.x)$;
5. Вычисляются точки плоскости, как пересечение линий $p_1 + t_1 * d_1$ и $p_2 + t_2 * d_2$:
   $vec2 \; C_1 = intersect(a_0.xy - n_0 * R, v_0, a_2.xy - n_1 * R, v_1)$,
   $vec2 \; C_2 = intersect(a_0.xy + n_0 * R, v_0, a_2.xy + n_1 * R, v_1)$,
   $vec2 \; C_3 = intersect(a_1.xy - n_1 * R, v_1, a_3.xy - n_2 * R, v_2)$,
   $vec2 \; C_4 = intersect(a_1.xy + n_1 * R, v_1, a_3.xy + n_2 * R, v_2)$,
   где функция пересечения $intersect$ определена следующим образом:
   $vec2 \; intersect(p_1, d_1, p_2, d_2)\{$
   $float \;\; t_1 = \frac{(p_1.x*d_2.y - p_1.y*d_2.x - p_2.x*d_2.y + p_2.y*d_2.x)}{(d_1.x*d_2.y - d_1.y*d_2.x)}$;
   $return \; p_1 + t_1 * d_1;\}$
6. Выполнятся проективные преобразования над полученными точками:
   $C_i = ModelViewMatrix * C_i \; for \; i \in [1:4]$.

После того, как для рассматриваемого сегмента найдены координаты плоскости, необходимо определить нормаль и позиции угловых точек pos в каждой точке интерполяции. Обозначим интерполированный радиус R через r. Используя вычисленные значения направления сегмента way и нормаль к отрезку сегмента n_1, во фрагментарном шейдере рассчитывается нормаль для каждого фрагмента следующим образом:

1. Вычисляется нормаль, направленная на наблюдателя:
   $vec2 \; n_{eye} = R * (vec3(n_1, 0) \times way)$;
2. Вычисляется интерполированный радиус в направлении нормали n_1:

$$vec2\ r_{side} = r * n_1$$

3. Вычисляется нормаль цилиндра в интерполированной точке:
   $vec2\ n_{cylinder} = normalize(r_{side} + n_{eye})$.
4. Рассчитывается цвет фрагмента методом Фонга, используя нормаль и $n_{cylinder}$ цвет траектории.

ЗАКЛЮЧЕНИЕ

В данной статье предложены новые подходы к визуализации структурированных сеточных моделей больших размерностей ($> 10^8$ ячеек), проведен сравнительный анализ подходов. Предложен алгоритм визуализации ребер ячеек сеточной модели с использованием геометрического шейдера, что позволяет визуализировать сетку вместе с визуализацией полигональной модели в один проход, без передачи дополнительных данных. Представлен алгоритм визуализации 3D трубок двухмерными объектами, которые выглядят как трехмерные цилиндры высокой степени детализации и при этом позволяют отображать большое количество сегментов без ощутимой нагрузки на графический и центральный процессоры. На основе предложенных подходов разработан прототип модуля визуализации.

СПИСОК ЛИТЕРАТУРЫ

1. Wolff D., OpenGL 4.0 Shading Language Cookbook. Packt Publishing Ltd. Birmingham (2011) 17. Wolfram, S.: Mathematica Book. [D. Wolff, OpenGL 4.0 Shading Language Cookbook. Packt Publishing Ltd. Birmingham (2011) 17. Wolfram, S.: Mathematica Book.]

2. Sellers G., Wright R.S. Jr., Haemel N. OpenGL SuperBible: Comprehensive Tutorial and Reference (6th Edition). Addison-Wesley Professional; 6 edition (July 31, 2013) 2013 P. 848. [G. Sellers, R.S. Jr. Wright, N. Haemel, OpenGL SuperBible: Comprehensive Tutorial and Reference (6th Edition). Addison-Wesley Professional; 6 edition (July 31, 2013) 2013 P. 848.]

3. Schlumberger. Eclipse reference manual: technical description, 2002 P. 2683. [**Schlumberger.** Eclipse reference manual: technical description, 2002 P. 2683.]

4. Merland R., Levi B., Caumon G. Building PEBI grids conforming to 3D geological features using centroidal Voronoi tessellations. // Proceedings IAMG. Salzburg (2011). 12 p. [R. Merland, B. Levi, G. Caumon**,** "Building PEBI grids conforming to 3D geological features using centroidal Voronoi tessellations", in Proc. IAMG. Salzburg (2011). 12 p.]

5. Aziz K., Settari A. Petroleum Reservoir Simulation, Calgary : K. Aziz & A. Settari 2002 p. 476. [K. Aziz, A. Settari, Petroleum Reservoir Simulation, Calgary : K. Aziz & A. Settari 2002 p. 476.]

6. Шурина Э. П., Войтович Т. В., Анализ алгоритмов методов конечных элементов и конечного объема на неортогональных сетках при решении уравнений Навье–Стокса, // Вычислительные технологии. 1997. Т. 2. №4. [E. P. Shaurina, T.V. Voitovich, Analysis of algorithms for finite element and finite volume on unstructured grids for solution of the Navier-Stokes equations, (in Russian). Computational technologies. 1997. T. 2. №4.]

7. Abraham F., Celes W. Distributed Visualization of Complex Black Oil Reservoir Models // Eurographics Symposium on Parallel Graphics and Visualization (2009), EGPGV 2009, Munich, Germany P. 87–94. [F. Abraham, W. Celes, **"**Distributed Visualization of Complex Black Oil Reservoir Models", Eurographics Symposium on Parallel Graphics and Visualization (2009), EGPGV 2009, Munich, Germany P. 87–94.]

8. Celes W., Abraham F. Texture-based wireframe rendering. // Brazil, Gramado, 23rd Graphics, Patterns and Images (SIBGRAPI) Conference, 2010, p. 149–155. [W. Celes, F. Abraham, "Texture-based wireframe rendering. Brazil, Gramado", 23rd Graphics, Patterns and Images (SIBGRAPI) Conference, 2010, p. 149–155.]

9. Two Methods for Antialiased Wireframe Drawing with Hidden Line Removal/ Bærentzen J. A. [et al.] // Proceedings of 24th Spring conference on computer graphics, SCCG '08, New York: ACM, 2008 P. 171–177. [J. A. Bærentzen, et al., "Two Methods for Antialiased Wireframe Drawing with Hidden Line Removal", in Proc. of 24th Spring conference on computer graphics, SCCG '08, New York: ACM, 2008 P. 171–177.]

10. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful/ Bittner J. [et al.] // Eurographics. 2004. Vol. 23, P. 615-624. [J. Bittner et al., "Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful", Eurographics. 2004. Vol. 23, P. 615-624.]

11. Mattausch O., Bittner J., Wimmer M. CHC++: Coherent Hierarchical Culling Revisited. // Eurographics. 2008. P. 221–230. [O. Mattausch, J. Bittner, M. Wimmer, "CHC++: Coherent Hierarchical Culling Revisited", Eurographics. 2008. P. 221–230.]

12. Tarini M., Cignoni P., Montani C. Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization. // IEEE Transactions on Visualization and Computer Graphics. V.12 №.5, P.1237-1244, September 2006. [M. Tarini, P. Cignoni, C. Montani, "Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization", IEEE Transactions on Visualization and Computer Graphics. V.12 №.5, P.1237-1244, September 2006.]

13. McKesson J. L. Learning Modern 3D Graphics Programming. [Электронный ресурс]. URL: https://web.archive.org/web/20150225192604/http://www.arcsynthesis.org/gltut/Illumination/Tutorial%2013.html#d0e12594 (дата обращения 07.11.2014). [J. L. McKesson. (2014, Nov. 7). Learning Modern 3D Graphics Programming. [Online]. Available: https://web. archive.org /web/20150225192604/http://www.arcsynthesis.org/gltut/Illumination/Tutorial%2013.html#d0e12594]

ОБ АВТОРАХ

БАДРЕТДИНОВ Марсель Рафаэлевич, дипл. мат-к-программист (УГАТУ, 2012), асп. факультета ИРТ, каф. ВМиК, Уфимский государственный авиационный технический университет (УГАТУ). Иссл. в обл. рекомендательных сист.

БАДРЕТДИНОВ Тимур Рафаэлевич, дипл. мат-к-программист (УГАТУ, 2012), асп. факультета ИРТ, каф. ВМиК, Уфимский государственный авиационный техниче-

кий университет (УГАТУ). Иссл. в обл. рекомендательных сист.

БОРЩУК Олег Сергеевич, дипл. мат-к-программист (УГАТУ, 2005). Зам. директора департамента инновационных технологий ООО «РН-УфаНИПИнефть». Иссл. в обл. численного моделирования фильтрации флюидов в пористой среде.

METADATA

Title: New technologies in visualization of geo-hydrodynamic models.

Authors: M.R. Badretdinov[1], T.R. Badretdinov[2], O.S. Borschuk[3].

Affiliation:
[1,2] Ufa State Aviation Technical University (UGATU), Russia.
[3] RN-«UfaNIPINeft», Россия.

Email: [1]BadretdinovMR@ufanipi.ru,
[2]BadretdinovTR@ufanipi.ru, [3] BorschukOS@ufanipi.ru

Language: Russian.

Source Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), vol. 19, no. 4 (70), pp. 84-94, 2015. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: In this paper we analyze problems of visualization of large geohydrodynamic models (HDM), consider deprecated and modern approaches in visualization of HDM. We suggest new approach for HDM visualization that is memory effective and allows to visualize large (>10^8 cells) HDM on PC with standard configuration. We present new approach in wireframe visualization of HDM using geometry shaders that allows rendering of a wireframe model in one draw call with polygonal model without additional data throughput from CPU to GPU. In this paper new algorithm for visualization of wells' trajectories is given that is based on imposter objects. It allows to draw large amount of well without exhausting cpu and gpu resources.

Key words: Computer graphics, reservoir visualization, geohydrodynamic visualization, wireframe rendering, geometry shader, imposter objects.

About authors:

BADRETDINOV Marsel Rafaelevich, Dipl. Mathematician-programmer (USATU, 2012), PhD student [Department of Computer Science and Robotics, Ufa State Aviation Technical University (USATU)], Ufa, Russia. Researcher in the field of recommender systems.

BADRETDINOV Timur Rafaelevich, Dipl. Mathematician-programmer (USATU, 2012), PhD student [Department of Computer Science and Robotics, Ufa State Aviation Technical University (USATU)], Ufa, Russia. Researcher in the field of recommender systems.

BORSCHUK Oleg Sergeyevich, Dipl. Mathematician-programmer (USATU, 2005). Deputy director of department for innovation technologies in RN-UfaNIPIneft company.