Список литературы

1. Технологии ГРИД в вычислительной химии / В.М. Волохов [и др.] // Вычислительные методы и программирование. – 2010. – Т. 11, № 1. – с. 42–49.

2. Вычислительная химия в грид-средах / В.М. Волохов [и др.] // Computational Chemistry in the Grid Environments // Distributed Computing and Grid-technologies in science and education: 4th Int. Conf. Dubna, JINR, 2010. P.143–144.

3. GRID и вычислительная химия / В.М. Волохов [и др.] // Вычислительные методы и программирование. – 2009. – Т. 10, № 2. – С.78–88.

4. Виртуальные вычислительные среды: использование на GRID полигонах / В.М. Волохов [и др.] // Вестн. ЮУрГУ. Серия «Математическое моделирование и программирование». – 2009. – № 17 (150). Вып. 3. – С. 24–35.

Н.И. Гаврилов, А.А. Белокаменская

Нижегородский государственный университет им. Н.И. Лобачевского

ОРГАНИЗАЦИЯ ПОТОКОВЫХ ВЫЧИСЛЕНИЙ НА GPU В ЗАДАЧЕ СТЕРЕОВИЗУАЛИЗАЦИИ ТОМОГРАММ

В научной визуализации часто приходится иметь дело со скалярными полями данных, заданными в трехмерном пространстве. Данные могут быть получены самыми различными способами – численным экспериментом, сканированием с помощью магнитного резонанса, компьютерной томографией (КТ), сканированием с помощью ультразвука. Например, результатом КТ является множество слоёв, т.е. двумерных массивов данных, которые вместе образуют трехмерное скалярное поле. Объем данных эксперимента значителен. Например, объем одного исследования в медицинской томографии обычно равен 0,25–1 Гб. Обеспечение реального времени визуализации означает вывод на экран порядка 25 и более кадров в секунду.

Метод прямого объёмного рендеринга методом обратной трассировки лучей с 90-х годов прошлого века позиционирует себя как эффективный инструмент для визуального анализа объёмных данных. Разработаны и доведены до широкой научной общественности [1] подходы, позволяющие решать задачи объемного рендеринга в реальном времени на основе параллельных вычислений и высокопроизводительной вычислительной техники. Эффективно решаются в интерактивном режиме задачи сегментации объема [2]. Развитие GPU как вычислительных устройств в последние годы дало этому направлению новый значительный импульс и позволило говорить о многообъемном рендеринге в реальном времени [3–5]. Стереовизуализация научных данных достаточно давно культивируется в Бостонском университете (http://scv.bu.edu), но не вошла пока в широкую практику в медицине.

Методы визуализации. В медицине томограммы хранятся в специальном формате данных DICOM, а для визуализации томограмм используют так называемые DICOM-визуализаторы. Довольно большая часть их общедоступна, в том числе: eFilm, MicroDicom, OsiriX, Onis-Viewer, RadiAnt и отечественный MultiVox DICOM Viewer (www.multivox.ru). Все эти Viewers обеспечивают визуализацию томограммы в виде классических полутоновых изображений 2D-проекций и сечений объёмного скалярного поля. Большинство Viewers обеспечивают также возможность визуализации 3D-моделей томограммы, используя чаще всего простые модификации прямого объемного рендеринга.

Наилучшего качества и информативности объёмного рендеринга позволяет достичь метод обратной трассировки лучей. Лучи обрабатываются независимо друг от друга, в алгоритме преобладают векторные операции, можно не делать большого количества ветвлений в алгоритме. Таким образом, GPU хорошо подходит для задачи объёмного рендеринга с использованием трассировки лучей. С помощью этого метода можно визуализировать данные как набор изоповерхностей или в виде тумана, или и то и другое вместе. У изоповерхностей и тумана можно варьировать прозрачность

и цвет. В научном эксперименте и дорогих профессиональных системах можно встретить примеры мультиобъемного рендеринга, к которому пока не предъявляют требования реального времени.

Высокое качество изображения формирует интерес к использованию стереоизображений, в которых так же, как в прямом рендеринге, используется цвет и прозрачность вокселей.

Реализация трассировки лучей для стереовизуализации объёмов на GPU. Наиболее удобной структурой хранения пространственных массивов данных на GPU является 3D-текстура, аналогичная 3D-массиву действительных чисел. Для CPU удобнее может оказаться иерархическая структура, например, октодерево. Для доступа к элементам дерева процессору необходимо пройти множество ветвлений, что приемлемо для CPU, но убивает производительность для GPU, к тому же в программах для GPU нет рекурсии. Программа, реализующая трассировку, выполнена в среде MS Visual Studio 2008, как шейдерная программа для GPU, в среде OpenGL и GLSL (шейдерный язык для OpenGL).

Предложен метод ускорения вычислений, заключающийся в использовании иерархии текстур: помимо основного массива 512×512×512 в GPU загружается меньший массив – двухканальная текстура 16×16×16. В пространстве каждому элементу этой текстуры соответствует свой блок 32×32×32 исходных данных. Текстура хранит минимальное и максимальное значения данных соответствующего блока.

Обращаясь к маленькой текстуре, луч может пропускать «неинтересные» области пространства, двигаясь с большим шагом. Около 20–30 % лучей так и не найдут «интересные» области, что иногда повышает FPS в три раза.

Различные алгоритмы трассировки луча. Для каждого пикселя искомого изображения генерируется луч, который проходит через объёмные данные с некоторым шагом. На каждом шаге луч может накапливать цвет. Таким образом, алгоритм накопления цвета определяет получаемое изображение. Рассмотрим алгоритм визуализации изоповерхности.

Визуализация изоповерхностей для объёмных данных – это довольно информативный визуальный анализ. Например, для визуализации костной ткани для данных CT-снимка достаточно построить изоповерхность для значения 500 (рентгеновская плотность кости по шкале Хаунсфилда +400 и выше). Цвет в соответствующем алгоритме накапливается лучом только при пересечении изоповерхности. Пересечение имеет место, если на соседних шагах луча выборки значений данных лежат по разные стороны изозначения. Далее численно вычисляем нормаль и определяем цвет, используя локальную модель освещения. Нормаль к изоповерхности определяется как нормированный градиент, который вычисляется численно, по разностной схеме. Помимо изоповерхностей можно также визуализировать контуры, проекции максимальной интенсивности и т.д.

Код программ для GPU. Каждый способ визуализации на практике представляет собой ту или иную шейдерную программу. Размер каждой из них составляет порядка 300 строк, 200 из которых составляют различные функции, одинаковые для всех алгоритмов. Среди таких функций – выборка данных в точке вычисления, в точке нормали, локального освещения, глубины и т.д. Остальные 100 строк занимает функция main, в которой и происходит трассировка луча из пикселя для вычисления цвета и глубины (для буфера глубины). Эта функция своя для каждого метода трассировки, поэтому загружаемая в GPU программа склеивается из заголовочной части, которая содержит функции и различные переменные, и основной части, где содержится функция main. Также во избежание дублирования кода без ущерба производительности происходит предварительная обработка текста шейдерной программы: например, вместо последовательности символов, начинающихся с ‘$’, подставляются конкретные значения. Ниже приведён пример: функция вычисления глубины, которая в зависимости от того, перспективная проекция используется или ортогональная, по-разному вычисляет глубину (pos – позиция наблюдателя, ps – позиция точки, глубину которой мы вычисляем, nav – ориентация камеры).

```
float GetDepth(vec3 ps)
{
      ps -= pos;
#if $IsPerspective==1
      return      clamp(z_far/(z_far-z_near)      -
(z_far*z_near/(z_far-
z_near))/(dot(ps,nav)),0.01,0.99);
#else
      return     clamp((dot(ps,nav)-z_near)/(z_far-
z_near),0.01,0.99);
#endif
}
```

Вычисление глубины изображения нужно для правильного вывода остальных объектов сцены: рамка ограничивающего объёма, секущие плоскости и т.д.

Ниже приведена часть шейдерной программы для визуализации непрозрачной изоповерхности. Здесь луч идёт с небольшим шагом step и на каждом шаге непосредственно обращается к исходным данным через функцию EquF. Выход из цикла происходит, либо если луч вышел за пределы ограничивающего объёма, либо при пересечении изоповерхности с изозначением min_level (эта часть кода для случая, когда изначально луч находился в точке, значение данных в которой было меньше min_level).

```
while(ps == clamp(ps,box1,box2))    //пока    внутри
объёма с данными
{
      e0=e;                         //храним  старое
значение выборки из данных
      e = EquF(ps);                 //выборка     из
объёмных данных

      if(e >= min_level)            //если
пересекли изоповерхность
      {
```

```
            ps    +=    step*((min_level-e)/(e-e0));
      //уточняем точку пересечения
            norm     =     -normalize(GradEqu(ps));
      //вычисляем нормаль
            color                                    =
Phong(ps,norm,vec3(0.7,0.7,0.7));
            gl_FragDepth       =       GetDepth(ps);
      //вычисляем глубину точки ps
            break;
      }
      ps+=step;          //движение луча
}
```

Функции PhongL и Phong вычисляют цвет в точке ps по модели освещения Фонга, однако первая функция к тому же увеличивает либо уменьшает яркость цвета в зависимости от того, заслонена или нет точка ps изоповерхностью. Замена простых условий if на препроцессорные #if нужны для уменьшения размеров скомпилированной шейдерной программы и ускорения самой компиляции. Для режима вывода множества полупрозрачных изоповерхностей используется массив их изозначений и оптических свойств (цвет и прозрачность).

Ещё пример: использование ускоряющей структуры. Перед циклом выше, где идёт накопление цвета, луч проходит через пустые блоки исходных данных. Обращение при этом идёт только к данным ускоряющей структуры, а шаг луча не фиксирован, чтобы пройти все ячейки ускоряющей структуры по пути трассировки.

```
#if $use_accel_struct==1      //если          хотим
использовать ускоряющую структуру
      for(float i=0.0;i<100.0;i++)
      {     //выходим  из  цикла,  если  вышли  за
            пределы     данных,     либо     нашли
            //“интересную” область
```

```
            if(!(ps   ==   clamp(ps,box1,box2))   ||
IEqu(ps)>=min_level)break;
            //шаг   луча   (переход   к   следующему
            пересечению      ячеек      ускоряющей
            //структуры)
            ps=GetCellStep(ps,ray);
      }
#endif
```

В целом использование такой оптимизации ускоряет рендеринг в 1,5–2,5 раза в зависимости от стоимости трассировки луча (визуализация контуров в 5–7 раз дороже визуализации непрозрачной изоповерхности). Пропуск пустых областей происходит только в начале – это оптимальное решение для GPU. Если искать пустые области и дальше, после прохода «интересных», то мы получим только проигрыш в производительности, поскольку постоянное обращение к ускоряющей структуре во время накопления цвета сведёт на нет выигрыш от повторного нахождения пустой области. На рисунке выигрыш от ускоряющей структуры составил 1,5, здесь лучи почти сразу начинают движение в «интересных» областях данных.

Ниже приведены результаты замера производительности (числа кадров в секунду) различных методов рендеринга на различных видеокартах. Информация о центральном процессоре отсутствует, поскольку CPU никак не влияет на производительность рендеринга.

– Размер окна вывода: 1280×1024 (во весь экран).

– Тестовые данные: 512×512×512, 16-битные.

– Длина шага луча: 0,0004 (0,2 от размера вокселя).

Ракурс крупным планом (см. рисунок, *а*), поэтому общее число шагов всех лучей оценивается величиной 1280·1024/0,0004 = = 3,2·10^9. Благодаря раннему прекращению движения луча из-за накопления достаточной непрозрачности это число уменьшается на порядок (таблица).

Результаты замера производительности

	Fog + Isos	Isos	MIP	Shaded Fog	Fog	Iso
NVIDIA GeForce GTS 250	9	10	10	12	12	30
NVIDIA GeForce 9500 GT	4	4	4	5	6	13
NVIDIA Quadro FX 5600	14	16	19	23	25	63
NVIDIA GeForce 8600 GT	3	4	2	4	4	9
ATI RADEON HD 4870	18	21	35	38	40	81
ATI RADEON HD 4890	21	25	39	44	49	108

В первой строке таблицы перечислены различные методы рендеринга:

Fog – туман (закраска пространства в соответствии с передаточной функцией (transfer function));

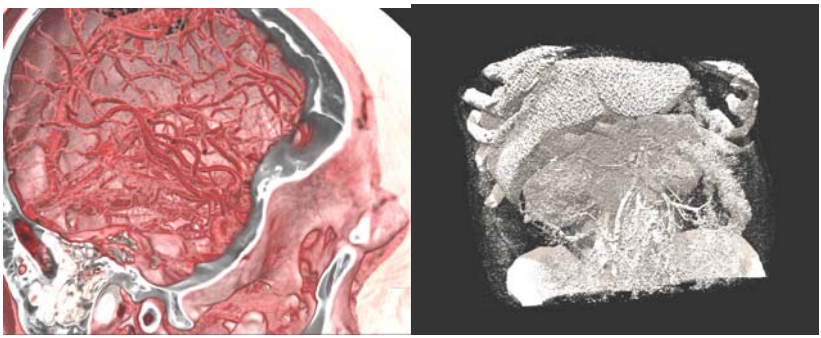
Isos – множество полупрозрачных изоповерхностей (в тесте их 3);

Fog + Isos – комбинация тумана и изоповерхностей;

Iso – непрозрачная изоповерхность;

MIP – проекция максимальной интенсивности;

Shaded Fog – затенённый туман (на цвет в точке влияет не только значение данных, но и градиент в данной точке).



а *б*

Рис. 3D-визуализация в программе SMV. Техники рендеринга: «затенённый туман» (*а*) и непрозрачная изоповерхность с тенью (*б*)

Из таблицы видно, что видеокарты ATI явно превосходят видеокарты NVIDIA в задаче трассировки лучей, что объясняет-

ся наличием большого числа векторных операций в алгоритмах, а видеокарты ATI как раз и ориентированы на выполнение векторных операций, тогда как ядра карт NVIDIA могут поодиночке выполнять скалярные операции.

Для сравнения производительности OpenCL и GLSL-шейдеров написаны две программы, реализующие алгоритм без ускоряющей структуры, рисующий одну изоповерхность для плотности 65 (плотность кости) и освещенный туман для данных всего диапазона плотностей. Протестировано на 32-битных данных размером 256^3. Для проведения эксперимента использовался графический ускоритель NVIDIA GeForce 8600GT и центральный процессор Intel Core2Duo 2,33 GHz. Размер окна визуализации составлял 512×512 точек. Производительность на OpenCL равна 5,6 fps, на GLSL – 10,2 fps. Таким образом, для некоторых задач реализация на шейдерах пока ещё остаётся оптимальным вариантом.

Авторы благодарны проф. В.Е.Турлапову за постановку задачи и участие в обсуждении результатов. Работа выполнена при финансовой поддержке ФЦП «Научные и научно-педагогические кадры инновационной России», госконтракт № 02.740.11.0839.

Список литературы

1. Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron E. Lefohn, Christof Rezk Salama, Daniel Weiskopf. Real-Time Volume Graphics, SIGGRAPH–2004. – URL: http://old.vrvis.at/via/resources/course-volgraphics-2004.

2. Построение 3D модели кровеносных сосудов по серии КТ-изображений печени / А.М. Ятченко [и др.]. Conf. Proc. of the 19th International Conference on Computer Graphics and Vision «GraphiCon'2009», Moscow, 2009.

3. Friedemann Roßler, Eduardo Tejada, Thomas Fangmeier, Thomas Ertl, Markus Knauff. GPU-based Multi-Volume Render-

ing for the Visualization of Functional Brain Images. – URL: http://www.vis.uni-stuttgart.de/ger/research/pub/pub2006/simvis06-roessler.pdf.

4. Flexible GPU-Based Multi-Volume Ray-Casting / R. Brecheisen [et al.] // Technical University of Eindhoven. – URL: http://www.yp. wtb.tue.nl/pdfs/9881.pdf

5. Beyer Johanna. GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery. – Dissertation. – Vienna University of Technology, okt. 2009. – 131 p.

6. The scientific computing and imagine institute at the university of Utah. – URL: http://www.sci.utah.edu/download/ IV3DData.html.

Р.К. Газизов, И.Р. Фатхулисламов

Уфимский государственный авиационный технический университет

GPU И MATLAB ДЛЯ РЕШЕНИЯ ЗАДАЧИ АВТОМАТИЧЕСКОЙ ГЕОГРАФИЧЕСКОЙ ПРИВЯЗКИ СПУТНИКОВЫХ ИЗОБРАЖЕНИЙ С ПИКСЕЛЬНОЙ ТОЧНОСТЬЮ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

В связи с резким увеличением количества графических карт для персональных компьютеров и рабочих станций графический процессор (GPU) стал платформой для параллельных вычислений, доступной для широкого круга пользователей. Учитывая его высокую пропускную способность памяти и высокую вычислительную производительность, GPU все чаще стали использовать для расчетов общего назначения (GPGPU). Множество удачных примеров использования GPU для реализации различных задач показаны в работе [1].

На сегодняшний день задача автоматической географической привязки спутниковых изображений является актуальной, так как в последнее время наблюдается устойчивая тенденция к увеличению объема получаемой информации из космоса и решение данной задачи требует значительного времени.