УДК 519.6

ОБРАБОТКА СЕТОЧНЫХ ДАННЫХ НА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

М. В. Якобовский
(ИММ РАН)

Рассматриваются алгоритмы и методы обработки на многопроцессорных системах данных, определенных на неструктурированных многомерных сетках большого размера. Обсуждаются алгоритмы измельчения пирамидальных сеток, рационального разбиения неструктурированных сеток, сжатия триангулированных поверхностей и визуализации трехмерных данных в распределенных системах.

Вычислительные системы, объединяющие сотни и тысячи процессоров, успешно используются для моделирования задач газовой динамики, горения, микроэлектроники, экологии и многих других. Их быстрое развитие привело к увеличению разрыва между возможностями традиционных последовательных средств подготовки и анализа данных и способностью многопроцессорных систем к генерации больших массивов результатов. Вероятно, следует уже на этапе постановки задачи стремиться к сокращению объема информации, предназначенной для непосредственного анализа человеком, и оперировать только ограниченным объемом некоторых интегральных характеристик, но необходимость проанализировать результаты расчета детально все же возникает. Подробный анализ сложно выполнить без привлечения средств визуализации, позволяющих увидеть как грубую картину, к примеру, газодинамического течения в целом, так и подробную структуру течения в ряде интересующих зон. Методы построения систем обработки и визуализации больших объемов сеточных данных имеют свою специфику, во многом определяемую особенностями вычислительных систем последнего поколения:

1. Вычислительной мощности любого отдельно взятого процессора не достаточно для обработки всего объема данных за приемлемое время.
2. Оперативной памяти любого отдельно взятого процессора не достаточно для хранения всего объема обрабатываемых данных.
3. Рабочие места большинства пользователей территориально удалены от крупных суперкомпьютерных центров. Связь между рабочими терминалами и вычислительными центрами обеспечивается относительно медленными каналами глобальных сетей, не позволяющими за разумное время передавать весь объем полученных результатов.

Использование триангуляций и тетраэдрических сеток, содержащих большое число узлов, позволяет с высокой точностью аппроксимировать пространство около двумерных и трехмерных тел сложной формы. Платой за увеличение точности описания геометрических объектов является не только увеличение по сравнению с регулярными сетками объема вычислений, приходящихся на каждый из узлов расчетной сетки, но и значительные трудности на этапах ввода/вывода данных, анализа результатов и др.

Многопроцессорные системы обладают производительностью, позволяющей использовать в расчетах достаточно подробные сетки и анализировать полученные на этих сетках результаты. Однако построение алгоритмов, в полной мере использующих возможности параллельных систем, является достаточно сложной задачей. С ростом числа узлов сетки значительно возрастает время, необходимое для решения задачи балансировки загрузки процессоров, чтения и записи данных о сетке и сеточных функциях. Эти затраты становятся сопоставимыми со временем отдельного сеанса моделирования.

Моделирование на многопроцессорных системах физических и технологических процессов с привлечением регулярных и нерегулярных многомерных сеток, содержащих порядка $10^8 \div 10^9$ узлов, требует создания параллельных методов для задач, которые решаются за приемлемое время последовательными методами:

- подготовки исходных данных: генерации и измельчения сетки;
- ввода/вывода данных: распределенного чтения, сжатия и записи сеточных данных;
- выполнения расчета: балансировки загрузки процессоров;
- анализа результатов: интерактивной удаленной визуализации сеточных данных.

Подготовка трехмерных пирамидальных сеток

Доступные в настоящее время генераторы не позволяют непосредственно получать трехмерные пирамидальные сетки большого размера. Сетка относительно небольшого размера (70 300 узлов), приведенная на рис. 1,*а*, подготовлена с помощью пакета, созданного в Институте вычислительной математики РАН. Данная сетка использовалась для моделирования обтекания сферы набегающим стационарным потоком.

Сетка равномерно сгущается к поверхности сферы и оси симметрии за сферой. На рис. 1,*а* показаны только треугольники сетки, лежащие на внешних границах расчетной области и непосредственно на сфере. После измельчения и корректировки с помощью упомянутого пакета фрагмента сетки, прилегающего к сфере, размер сетки этого фрагмента был увеличен примерно в 8 раз. Дальнейшее увеличение размера сетки возможно с помощью алгоритма измельчения, гарантирующего не более чем ограниченное, не зависящее от степени измельчения снижение уровня качества сетки.

Качество сетки определяется геометрическими свойствами образующих сетку примитивов. Количественно его можно определить величиной взятого по всем пирамидам сетки максимума отношения самого длинного ребра к самому короткому в каждой из пирамид. Чем ближе к единице эта величина, тем более "правильные" пирамиды образуют сетку.

Равномерное измельчение двумерной сетки не представляет проблемы (рис. 2), поскольку разбиение каждого треугольника на четыре, подобных исходному, решает задачу увеличения числа узлов при сохранении геометрических свойств сетки (например, при таком измельчении не меняется отношение длин минимального и максимального ребер в треугольниках сетки).

В трехмерном случае аналогичный алгоритм не известен, тем не менее равномерное измельчение пирамидальной сетки может быть выполнено следующим образом:

1. Каждая из пирамид разбивается на четыре подобных исходной пирамиде тетраэдра и один октаэдр. Вершинами вновь образованных многогранников являются вершины и середины ребер исходной пирамиды (рис. 3,*а*).
2. Каждая из образованных пирамид измельчается так же, как на шаге 1, а каждый из октаэдров разбивается на шесть октаэдров, подобных исходному, и восемь тетраэдров, подобных исходной пирамиде (рис. 3,*б*).

   Шаги 1 и 2 повторяются необходимое число раз до получения сетки заданного размера (требуемой точности).
3. Каждый из октаэдров разбивается на пирамиды, например на четыре (рис. 3,*в*). Этот шаг выполняется только один раз.

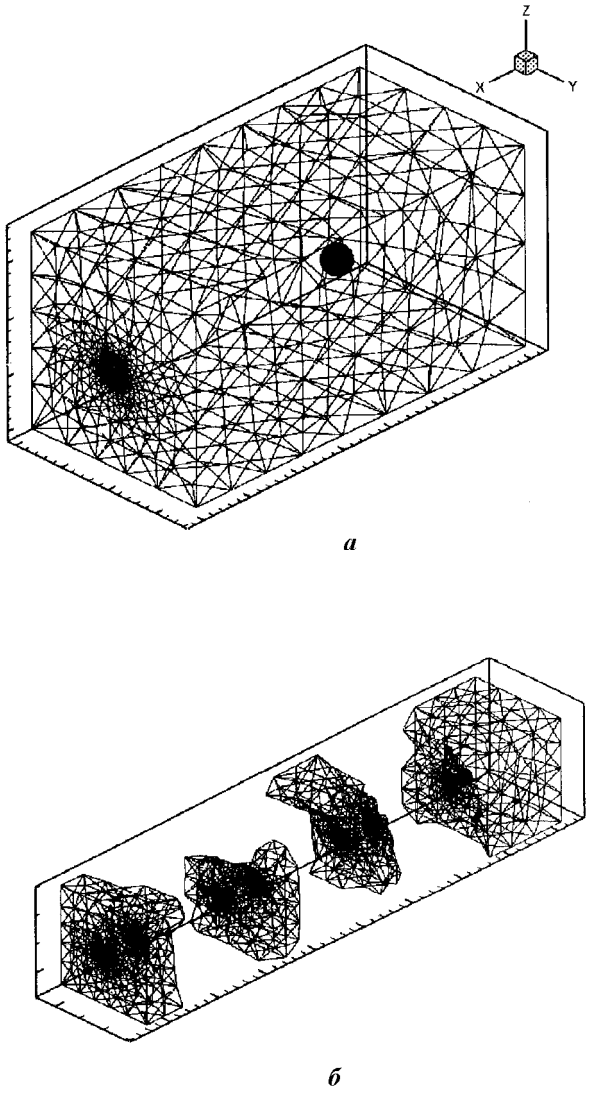


Рис. 1. Пирамидальная сетка (*а*) и ее разбиение на 4 части (*б*): 70 300 узлов, 401 418 тетраэдров, 1 336 881 ребер

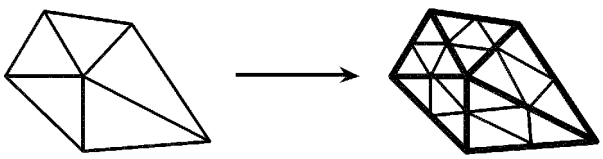


Рис. 2. Равномерное измельчение двумерной сетки

На первых двух шагах качество сетки не ухудшается, что позволяет повторить их произвольное число раз. Только на шаге 3, в момент разбиения октаэдров на пирамиды, происходит ухудшение качества сетки. Поскольку шаг 3 выполняется только один раз, итоговое ухудшение качества сетки не зависит от степени ее измельчения.

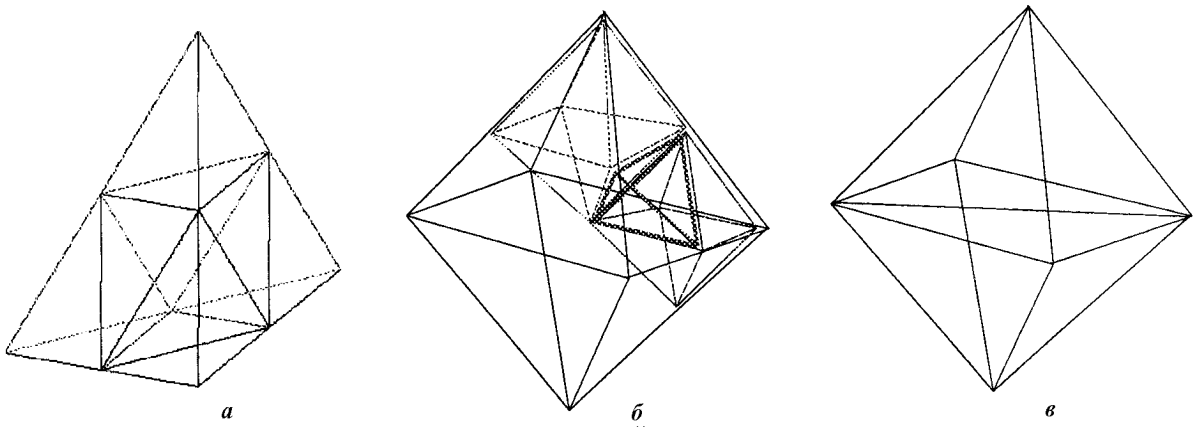


Рис. 3. Равномерное измельчение пирамидальной сетки: *а* — шаг 1; *б* — шаг 2; *в* — шаг 3

Декомпозиция сеток

Одной из основных проблем построения эффективных параллельных алгоритмов является задача балансировки загрузки процессоров — выбора такого распределения вычислительной нагрузки между процессорами, при котором их простои минимальны. При построении алгоритмов, предназначенных для вычислительных систем с раздельной памятью, необходимо, с одной стороны, равномерно распределить между процессорами суммарный объем вычислений, а с другой стороны, — минимизировать время, затрачиваемое на синхронизацию и передачу данных между процессорами.

Широко используемый при численном решении задач математической физики метод геометрического параллелизма предполагает деление сетки, покрывающей расчетную область, на множество доменов, каждый из которых обрабатывается отдельным процессором системы. При таком подходе по доменам распределяются сеточные узлы. Общая вычислительная нагрузка каждого из процессоров определяется суммарным временем обработки входящих в домен узлов сетки. Объем передаваемых между процессорами данных определяется весом ребер, соединяющих вершины, принадлежащие разным доменам. Можно полагать, что коммуникационная среда обеспечивает одновременную передачу данных между независимыми парами процессоров без снижения скорости передачи по сравнению с монопольным использованием коммуникационного оборудования парой процессоров. Таким образом, следует минимизировать не общий объем передаваемых в ходе расчета данных, а максимальный объем данных, передаваемых каждым из процессоров. Кроме того, полезно минимизировать число актов обмена данными, а значит, уменьшить число соседей у каждого из доменов.

При использовании регулярных сеток, топологически эквивалентных индексному прямоугольнику или индексному параллелепипеду, задача балансировки загрузки решается очевидным образом — прямым разбиением области на заданное число доменов, в каждом из которых содержится одинаковое число узлов, перпендикулярными к индексным осям гиперплоскостями.

Задача рационального распределения по процессорам нерегулярных сеток может быть сведена к разбиению на компактные домены вершин графа, веса узлов которого отражают объемы вычислений в различных расчетных узлах, а веса ребер — объемы обменов данными, выполняемых в процессе счета между соответствующими узлами. Для разбиения на домены нерегулярных сеток наиболее эффективны иерархические методы [1, 2], основой которых является следующая последовательность действий:

1. *Огрубление графа* — построение последовательности уменьшающихся в размере вложенных графов.
2. *Рекурсивная бисекция* — распределение вершин огрубленных графов по заданному числу доменов.
3. *Восстановление графа и локальное уточнение* — последовательное распределение по доменам вершин всех графов сформированной последовательности, начиная с имеющих наименьший размер, вплоть до исходного.

Пусть задан взвешенный граф $G^0 = (V, E)$, $V = \{v_i\}$, $|V| = n$, вершины v_i и ребра e_{ij} которого имеют веса $w(v_i)$ и $w(v_i, v_j)$ соответственно, причем $w(v_i, v_i) = 0$. Требуется найти такое разбиение $R(V) = (V_1, \ldots, V_p)$ вершин на заданное число доменов p

$$V = \bigcup_{k=1}^{p} V_k, \qquad V_i \bigcap V_j = \emptyset, \qquad i \neq j, \tag{1}$$

при котором достигается

$$\min_{R(V)} \left\{ J = \max_{k=1,\ldots,p} \sum_{v_i \in V_k} \left(w(v_i) + \alpha \sum_{v_j \notin V_k} w(v_i, v_j) \right) \right\}. \tag{2}$$

Мультипликативная константа α обеспечивает согласование единиц измерения весов вершин и ребер, что

позволяет формализовать требования выравнивания вычислительной нагрузки и снижения коммуникационной нагрузки на процессоры, сводя задачу к минимизации J.

Задача (1), (2) принадлежит классу NP, и в общем случае ее точное решение за обозримое время не представляется возможным уже при числе вершин графа, измеряемом сотнями, что заставляет использовать эвристические алгоритмы.

Огрубление графа. Алгоритм огрубления графа (рис. 4) основан на идее многократного стягивания ребер. В соответствии с ним из всего множества ребер графа G^k выбирается подмножество ребер покрытия таким образом, чтобы никакая из вершин не была инцидентна более чем одному из них. Далее каждая пара вершин $\left(v_i^k, v_j^k\right)$, соединяемых ребром покрытия e_{ij}^k, стягивается в одну агрегированную вершину v_t^{k+1}. Будем в дальнейшем называть вершины v_i^k, v_j^k *порождающими* для вершины v_t^{k+1} и записывать этот факт следующим образом: $v_t^{k+1} = \left\langle v_i^k \oplus v_j^k \right\rangle$. Некоторые из вершин G^k могут не иметь инцидентных им ребер покрытия; такие вершины тем не менее порождают вершины графа G^{k+1}: $v_t^{k+1} = \left\langle v_i^k \right\rangle$.

Пусть $v_{t1}^{k+1} = \left\langle v_{i1}^k \oplus v_{j1}^k \right\rangle$ и $v_{t2}^{k+1} = \left\langle v_{i2}^k \oplus v_{j2}^k \right\rangle$, тогда веса вершин и ребер графа G^{k+1} определяются следующим образом (рис. 4,*а*):

$$
\begin{aligned}
&w\left(v_{t1}^{k+1}\right) = w\left(v_{i1}^k\right) + w\left(v_{j1}^k\right);\\
&w\left(v_{t2}^{k+1}\right) = w\left(v_{i2}^k\right) + w\left(v_{j2}^k\right);\\
&w\left(v_{t1}^{k+1}, v_{t2}^{k+1}\right) = w\left(v_{i1}^k, v_{i2}^k\right) + w\left(v_{i1}^k, v_{j2}^k\right) +\\
&\quad + w\left(v_{j1}^k, v_{i2}^k\right) + w\left(v_{j1}^k, v_{j2}^k\right).
\end{aligned}
$$

Аналогично, если $v_{t1}^{k+1} = \left\langle v_{i1}^k \oplus v_{j1}^k \right\rangle$ и $v_{t2}^{k+1} = \left\langle v_{i2}^k \right\rangle$, то (рис. 4,*б*)

$$
\begin{aligned}
&w\left(v_{t1}^{k+1}\right) = w\left(v_{i1}^k\right) + w\left(v_{j1}^k\right);\\
&w\left(v_{t2}^{k+1}\right) = w\left(v_{i2}^k\right);\\
&w\left(v_{t1}^{k+1}, v_{t2}^{k+1}\right) = w\left(v_{i1}^k, v_{i2}^k\right) + w\left(v_{j1}^k, v_{i2}^k\right).
\end{aligned}
$$

Однократное выполнение описанной процедуры позволяет примерно вдвое уменьшить число вершин: $|V_{k+1}| \approx |V_k|/2$ (рис. 4,*в*).

В результате многократного огрубления формируется цепочка графов $G^k = \left(V^k, E^k\right)$: $\left|V^{k-1}\right| > \left|V^k\right|$, $k = 1, \ldots, m$. Разбиение на домены начинается с последнего из них (G^m). При выборе множества ребер покрытия графа G^k следует отдавать предпочтение ребрам, инцидентным вершинам, имеющим наименьшие веса, что позволяет уменьшить разброс весов вершин графа G^{k+1} и не только улучшить качество итогового разбиения, но и сократить время его построения.

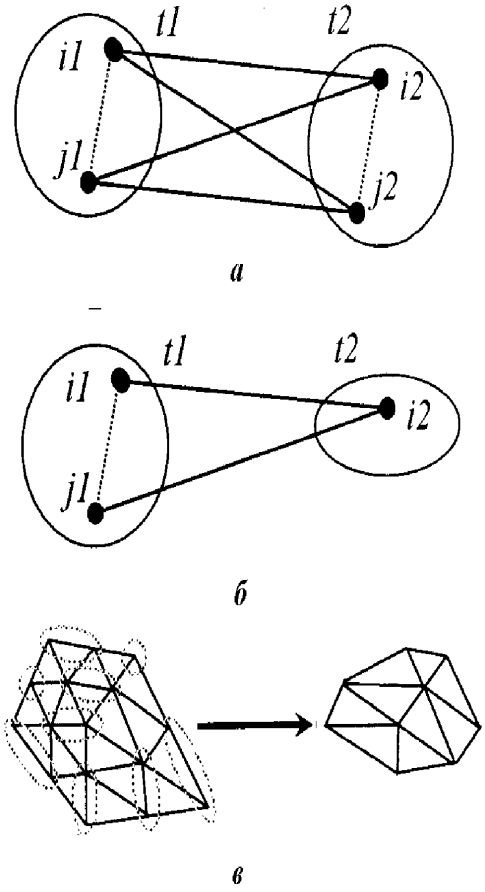


Рис. 4. Огрубление графа: *а* — случай 1; *б* — случай 2; *в* — результат огрубления

Разбиение огрубленного графа. Одним из наиболее привлекательных методов, пригодных для разбиения достаточно больших графов, является метод спектральной бисекции [2, 3]. Описываемый далее вариант этого метода выполняет рациональное разбиение графа на две компактные части, что позволяет, используя алгоритм рекурсивной бисекции, разбить граф на произвольное число частей.

1. *Рекурсивная бисекция.* Процедура рекурсивной бисекции (рис. 5) служит для разбиения графа, содержащего n вершин, на произвольное число доменов k. На рис. 6 представлен результат разбиения 100 вершин графа на 7 частей приблизительно равного размера. В каждом из корней дерева разрезов указано число вершин соответствующего подграфа; два числа, разделенные знаком косой черты, указывают пропорцию, в которой вершины подграфа разбиваются на две части.

2. *Спектральная бисекция.* Метод разбиения графа на два компактных домена в заданном отношении весов, минимизирующий суммарный вес ребер, соединяющих вершины из разных доменов, основан на использовании спектральной матрицы графа (матри-

```
bisect(graf,n, k1,k2) // k=k1+k2
{
n1= n*k1/(k1+k2);
n2= n-n1;

// разбиение вершин графа на две части
// размером n1 и n2

spectral(graf,n, &subgraf1,&n1, &subgraf2,&n2);

if(k1>1) bisect(subgraf1, n1, (k1+1)/2, k1/2);
if(k2>1) bisect(subgraf2, n2, (k2+1)/2, k2/2);
}
```

Рис. 5. Рекурсивная бисекция

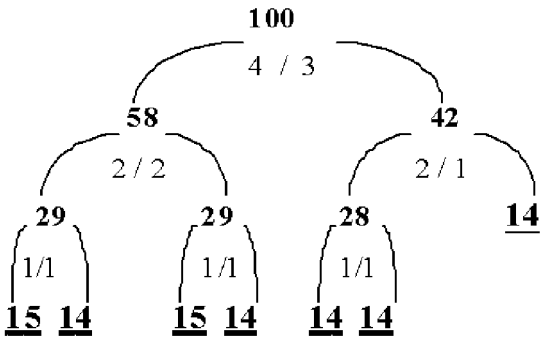
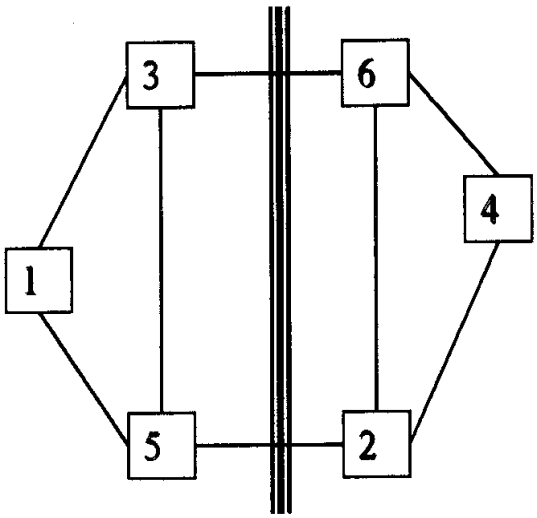


Рис. 6. Бинарное дерево разрезов



$$A = \begin{pmatrix} -2 & 0 & 1 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 & 1 & 1 \\ 1 & 0 & -3 & 0 & 1 & 1 \\ 0 & 1 & 0 & -2 & 0 & 1 \\ 1 & 1 & 1 & 0 & -3 & 0 \\ 0 & 1 & 1 & 1 & 0 & -3 \end{pmatrix}$$

Рис. 7. Граф и соответствующая ему спектральная матрица A

цы Лапласа) [3, 4] и описывается следующими этапами A—D.

A. Построение спектральной матрицы графа.
Элементы спектральной матрицы A графа G определяются соотношениями

$$a_{ij} = \begin{cases} w(v_i, v_j), & i \neq j; \\ -\sum\limits_{k=1,N}^{N} w(v_i, v_k), & i = j. \end{cases}$$

Рис. 7 иллюстрирует построение матрицы для графа, содержащего 6 вершин.

B. Определение первого собственного вектора.
Решение системы $Ax = \lambda x$ дает собственные значения и собственные векторы спектральной матрицы. Собственные значения матрицы на рис. 7 равны $\lambda_{1\ldots6} = \{0, -1, -3, -3, -4, -5\}$. Интерес представляет собственный вектор, соответствующий максимальному ненулевому собственному значению (вектор Фидлера [5]). В данном случае это вектор $\mathbf{x}_2 = (2, -1, 1, -2, 1, -1)$, соответствующий значению $\lambda_2 = -1$.

C. Упорядочение вершин графа.
Далее следует упорядочить множество номеров вершин графа по неубыванию значений компонент вектора Фидлера — определить вектор $\mathbf{s}$, удовлетворяющий условию $\mathbf{x}_2(s_i) \leq \mathbf{x}_2(s_j)$ при $i < j$. В рассматриваемом примере порядок вершин будет следующий: $\mathbf{s} = (4, 2, 6, 3, 5, 1)$.

D. Формирование двух доменов.
Используя вектор $\mathbf{s}$, можно распределить вершины графа на две части в требуемой пропорции $Q = n_1/n_2$. Первая часть должна содержать вершины, имеющие меньшие значения компонент найденного собственного вектора, вторая — имеющие большие значения. Точнее говоря, следует определить такой номер k, для которого выполняются условия

$$\sum_{i=1}^{k} w(v_{s_i}) \leq Q\,|V| \sum_{i=1}^{|V|} w(v_i);$$
$$\sum_{i=1}^{k+1} w(v_{s_i}) > Q\,|V| \sum_{i=1}^{|V|} w(v_i),$$

и включить в первый домен вершины, имеющие номера s_i, $i = 1, \ldots, k$, а во второй домен — остальные.

В рассматриваемом примере в первый домен попадают вершины $(4, 2, 6)$, а во второй — $(3, 5, 1)$, что соответствует оптимальному разбиению графа, изображенного на рис. 7, на две равные части.

Одна из частей может не быть связной, в этом случае для выполнения дальнейшего разбиения следует дополнить ее фиктивными ребрами, удалить которые можно после получения окончательного результата.

Найденное разбиение, как правило, является достаточно хорошим приближением к оптимальному решению, тем не менее можно улучшить его с помощью алгоритма локального уточнения.

Восстановление графа. После завершения этапов огрубления и бисекции получено разбиение графа G^k на p доменов: $R\left(V^k\right) = \left(V_1^k, \ldots, V_p^k\right)$. В качестве начального разбиения вершин G^{k-1} выберем такое $R\left(V^{k-1}\right) = \left(V_1^{k-1}, \ldots, V_p^{k-1}\right)$, в котором $v_t^{k-1} \in V_r^{k-1}$ тогда и только тогда, когда $v_q^k \in V_r^k$ и либо $v_q^k = \left\langle v_t^{k-1} \oplus v_j^{k-1} \right\rangle$, либо $v_q^k = \left\langle v_t^{k-1} \right\rangle$. Иными словами, порождающие вершины V^{k-1} распределяются в домены с теми же номерами, что и образованные ими вершины V^k. Полученное начальное разбиение можно существенно улучшить частичным перераспределением вершин между доменами с помощью алгоритма локального уточнения.

Локальное уточнение. При разбиении огрубленного графа практически неизбежно формируются домены несовпадающих весов, так как веса агрегированных вершин могут иметь значительный разброс. Локальное уточнение позволяет выровнять между собой веса доменов и уменьшить число ребер, пересекающих границы доменов (рис. 8).

Обозначим через $P(i)$ номер домена, которому принадлежит вершина v_i. Алгоритм является эвристическим и позволяет избежать полного перебора вариантов размещения вершин графа по доменам. С его помощью можно проверить выигрыш от перемещения каждой вершины v_i в каждый из доменов, соседних с доменом $P(i)$ (домены r и t называются *соседними*, если $\exists e_{ij} \in E : v_i \in V_r \; v_j \in V_t$), и выполнить перемещения, минимизирующие J (2). Выигрыш от перемещения определяется величиной уменьшения значения J. Каждая итерация алгоритма выглядит следующим образом:

1) для каждой вершины v_i определяется выигрыш $g_i(k)$, достигаемый в результате перемещения вершины i в домен k:

$$\overline{W} = \frac{\sum\limits_{v \in V} w_i}{|V|};$$

$$g_i^0(k) = \left|W_{P(i)} - \overline{W}\right| - \left|W_{P(i)} - \overline{W} - w_i\right| + \left|W_k - \overline{W}\right| - \left|W_k - \overline{W} + w_i\right|;$$

$$g_i^1(k) = \sum_{e_{ij} \in E} \begin{cases} w_{ij}, & P(j) = k; \\ -w_{ij}, & P(j) = P(i); \\ 0 & ; \end{cases}$$

$$g_i(k) = g_i^0(k) + g_i^1(k);$$

2) определяется некоторое множество пар (i, k), для которых $g_i^1(k)$ имеют большие значения;
3) для выбранных пар проверяется условие $g_i(k) > 0$ и, если оно удовлетворяется, вершина i включается в множество перемещаемых на данной итерации вершин H;
4) выполняется перемещение вершин выбранного множества H;
5) если J уменьшилось, то полученное разбиение принимается в качестве текущего и выполняется следующая итерация;
6) если уменьшения J не произошло, то сделанные перемещения отменяются и выбирается другое множество перемещаемых вершин или, если несколько попыток не увенчалось успехом, оптимальное разбиение считается найденным.

При определении величин $g_i(k)$ учитывается эффект от перемещения только одной вершины, но сумма выигрышей, полученных от перемещения двух вершин в отдельности, не равна выигрышу от их одновременного переноса. На рис. 8,*а* приведен пример графа, разделенного на два домена. Перенос из домена в домен любой из вершин v_2, v_3, v_4, v_5 уменьшает на единицу общее число ребер, пересекающих границу между доменами (рис. 8,*б*). Но одновременный перенос всех четырех вершин приводит к увеличению общего числа таких ребер с четырех до шести, снижая, таким образом, качество разбиения (рис. 8,*в*). Возможен и обратный эффект — положительный выигрыш при одновременном переносе нескольких вершин, даже если перенос некоторых из них в отдельности ведет к проигрышу. В связи с этим целесообразно допустить перенос некоторых вершин, частично ухудшающих разбиение.

Перенос каждой вершины приводит к изменению выигрышей от последующих перемещений ее самой и некоторых других вершин. Прямой пересчет всех выигрышей занимает значительное время. На первой итерации соответствующие потери неизбежны, но на последующих итерациях можно заменить полный пересчет выигрышей корректировкой уже вычисленных величин. Основное время требуется для расчета $g_i^1(k)$, описывающих выигрыш от изменения числа ребер, соединяющих домены. Общее число необходимых для этого действий оценивается как $T^0 = O\left(p\overline{\rho(v)}\,|V|\right)$, где $\overline{\rho(v)}$ — средняя степень вершин (среднее число инцидентных вершинам ребер). В реальных расчетных сетках средняя степень вершин не зависит от их общего числа ($\overline{\rho(v)} = O(1)$), таким образом, $T^0 = O(p|V|)$. Именно на

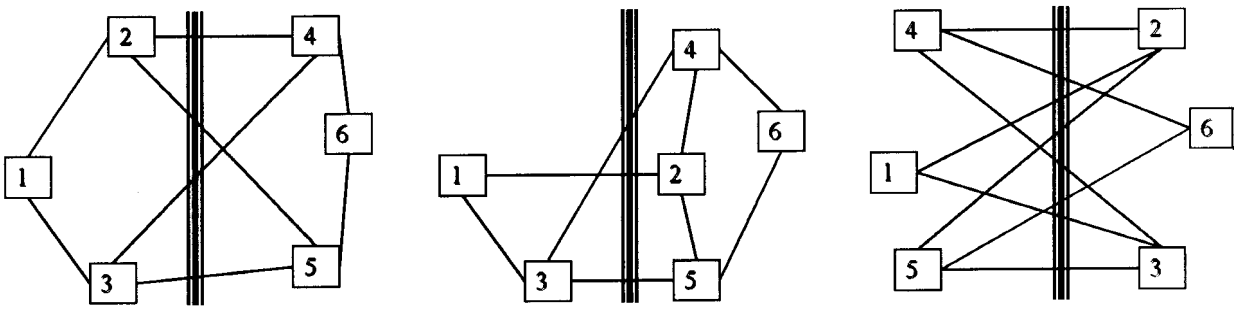


Рис. 8. Перенос вершин при локальном уточнении границ доменов: *а* — исходный граф, разделенный на два домена; *б* — перенос одной вершины из одного домена в другой; *в* — перенос четырех вершин

основе значений $g_i^1(k)$ происходит предварительный отбор пар (i,k), определяющих H. Величины $g_i^0(k)$ и $g_i(k)$ для большинства вершин не вычисляются совсем. При переносе одной вершины v_i число ребер, пересекающих границы доменов, может измениться только за счет ребер, инцидентных v_i, и, следовательно, обновить следует только элементы $g_t^1(k)$, $t \in GH$, соответствующие множеству вершин, соседних с вершинами H. Как правило, $|GH| \ll |V|$, поэтому время перерасчета выигрышей значительно меньше времени полного расчета всех элементов $g_i^1(k)$.

Для эффективной реализации изложенной идеи следует использовать такую дисциплину хранения $g_i^1(k)$, которая не только предоставит быстрый доступ к изменяемым элементам, но и даст возможность быстрого выбора претендентов на перемещение — вершин, обладающих максимальными выигрышами. Для хранения $g_i^1(k)$ можно использовать структуры данных типа *корзинка* B_{ij} [6]. Потребуется $p^2 - p$ корзинок (p — число доменов) — по числу возможных перемещений вершины из одного домена в другой. Каждая корзинка содержит $2g_{\max} + 1$ сбалансированных деревьев [7, 8] (AVL-деревьев) с номерами вершин домена ($g_{\max}$ — максимально возможный для данного графа выигрыш от перемещения одной вершины). В дереве B_{ij}^k хранятся номера вершин, перенос которых из домена i в домен j даст выигрыш k. Для каждой вершины хранится информация только об одном перемещении, соответствующем максимальному выигрышу от перемещения этой вершины. Следовательно, общее число записей в корзинке B_{ij} равно $|V_i|$. Сбалансированные деревья обеспечивают возможность добавления, поиска и удаления своих вершин за время порядка $O(\log M)$, где M — число вершин в дереве, что необходимо для сокращения времени выполнения основных операций:

- выбора вершин, перенос которых приносит максимальный выигрыш;
- перемещения вершин между корзинками;
- определения позиции в корзинке и обновления значений элементов $g_i^1(k)$.

Одновременно с заполнением корзинок следует поддерживать указатель на дерево вершин с максимальными для данной корзинки выигрышами, что позволит выполнить выбор каждой из них за время $T_1 = O(1)$.

При удалении вершины ее положение внутри корзинок определено заранее (на этапе формирования множества H) и выполнять поиск не требуется, следовательно, число действий определяется операцией удаления вершины сбалансированного дерева и оценивается как $T_2 = O\left(\log\left|B_{ij}^k\right|\right)$.

При добавлении вершины в корзинку номер требуемого списка соответствует предполагаемому выигрышу, а вставка вершины в сбалансированное дерево требует выполнения порядка $T_3 = O\left(\log\left|B_{ij}^k\right|\right)$ действий.

Обновление величин $g_i^1(k)$ при перемещении одной вершины требует выполнения порядка $T_4 = O\left(\log\left|B_{ij}^k\right|\right)$ действий, необходимых для перемещения вершин между сбалансированными деревьями, поскольку время вычисления величины, на которую изменяется выигрыш, составляет $O(1)$ и им можно пренебречь.

Таким образом, время выполнения операций перемещения вершин и обновления величин $g_i^1(k)$ для всего множества вершин H оценивается как $T^1 = |H|(T_1 + T_2 + T_3 + T_4) = O\left(|H|\log\left|B_{ij}^k\right|\right)$. Для планарных сеток T^1 не превышает $O(|V|)$, даже если перемещению подлежат все вершины, лежащие на границах доменов, и все они в каждой корзинке сосредоточены в одном сбалансированном дереве. Число вершин, лежащих на границе доменов, можно оценить как $|H| = O\left(p\sqrt{\frac{|V|}{p}}\right)$, поскольку предварительное разбиение к моменту применения алгоритма локального уточнения уже выполнено и домены имеют не оптимальную, но уже компактную форму. Таким образом, $T^1 = O\left(\sqrt{p|V|}\log\frac{|V|}{p}\right)$.

Общее время выполнения алгоритма можно оценить величиной

$$T = T^0 + T^1 = O\left(p|V| + \gamma\sqrt{p|V|}\log\frac{|V|}{p}\right),$$

где γ — число итераций алгоритма локального уточнения.

Величина γ, значительно влияющая на общее время работы алгоритма, косвенно зависит от выбранной стратегии перемещения вершин и определяется рядом эмпирических правил и параметров, например таких:

- каждая вершина переносится из одного домена в другой не более одного раза;
- каждую вершину из домена i можно переносить только в те домены, в которых существуют вершины, связанные ребрами с вершинами домена i;
- на каждой итерации переносится не более k вершин, где k выбирается эмпирически, например $k = |V|/10$.

Описанные алгоритмы успешно применимы к разбиению не только двумерных, но и трехмерных пирамидальных сеток. Пример разбиения на 4 домена сетки, используемой для расчета внешнего обтекания сферы, приведен на рис. 1,*б*.

Описание параллельного аналога изложенного последовательного алгоритма декомпозиции сеток выходит за рамки настоящей статьи.

Балансировка загрузки

При использовании многопроцессорных систем характерно выполнение длительных вычислений с помощью большого количества сравнительно коротких сеансов. Список используемых процессоров и их количество может меняться от одного сеанса к другому, в результате чего следует каждый раз заново решать задачу статической балансировки загрузки для определения оптимального распределения сетки по процессорам. Несмотря на высокую эффективность иерархических алгоритмов, разбиение нерегулярных расчетных сеток большого размера занимает значительное время. Помимо этого, значительное время занимает загрузка начальных данных и запись результатов расчетов, что само по себе заметно снижает эффективность отдельного вычислительного сеанса.

Для уменьшения указанных потерь целесообразно использовать иерархический метод хранения и обработки больших сеток. В соответствии с ним расчетная сетка предварительно разбивается на множество блоков небольшого размера — микродоменов (рис. 9) — и хранится в виде набора этих блоков и графа, определяющего связи блоков между собой. Такой способ представления сетки несколько увеличивает суммарный объем хранимой информации за счет взаимного наложения микродоменов, но позволяет избежать чтения и анализа всей сетки при определении параметров какой-либо подобласти. В результате предварительного разбиения сетки порождается новый граф, вершины которого соответствуют микродоменам (макрограф). Макрограф описывает топологию сетки подобластей, каждая из его вершин соответствует микродомену, содержащему множество узлов.

Перед очередным сеансом расчета производится разбиение макрографа на равное числу процессоров количество доменов, в результате чего каждый процессор получает список микродоменов, совокупность которых определяет обрабатываемую им часть расчетной области (см. рис. 9). Таким образом, опосредованно осуществляется распределение исходной сетки по процессорам, однако при этом разбиению непосредственно перед выполнением расчета подвергается макрограф, имеющий значительно меньшие размеры, чем исходная сетка. Разбиение макрографа требует небольшого по сравнению с разбиением исходной сетки времени и может быть выполнено при помощи рассмотренных последовательных алгоритмов. Для обеспечения приемлемого качества распределения узлов исходной сетки по процессорам желательно, чтобы количество микродоменов на 2—3 порядка превышало максимальное число процессоров, доступных для проведения расчетов. Данный подход позволяет выполнять на каждом процессоре чтение только обрабатываемых на нем микродоменов, что значительно сокращает необходимое для ввода/вывода время.

Визуализация

Как уже отмечалось, начиная с некоторого размера сетки для ее обработки недостаточно ресурсов однопроцессорной вычислительной системы рабочего места пользователя. Это обусловлено, во-первых, недостатком оперативной памяти; во-вторых, недостаточной вычислительной мощностью; в-третьих, ограниченностью пропускной способности каналов связи между терминалом пользователя и суперкомпьютерным центром.

Обойти эти ограничения можно, положив в основу системы визуализации модель клиент/сервер [9]. Разделение системы на две части, связанные относительно "тонким" (обладающим низкой пропускной способностью) каналом связи (рис. 10), позволяет получить преимущества по крайней мере по двум направлениям:

1) сервер визуализации может выполняться на многопроцессорной системе, используя необходимое число процессоров, обладающих в сумме требуемой вычислительной мощностью и оперативной памятью;
2) клиент визуализации может выполняться на персональном компьютере пользователя и использовать значительные аппаратные и программные мультимедийные возможности для построения наглядных визуальных образов и управления ими (графические ускорители, стерео-устройства, 2, 3, 6-мерные манипуляторы).

Таким образом, суперкомпьютер обеспечивает обработку и сжатие исходных данных, после чего

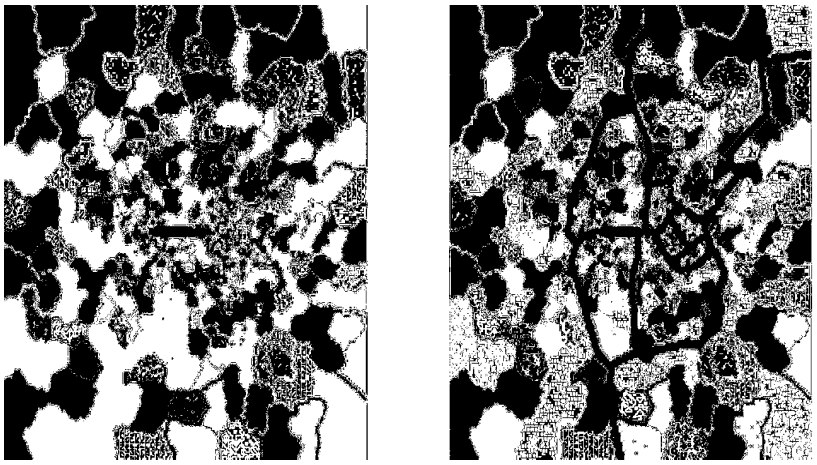


Рис. 9. Измельчение двумерной сетки

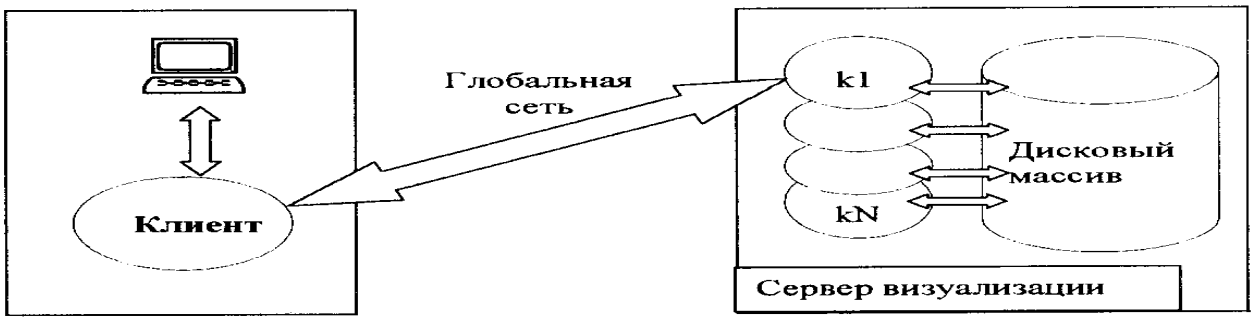


Рис. 10. Структура системы визуализации

небольшое количество описывающих изображение элементов передается на персональный компьютер. На их основе компьютер рабочего места пользователя обеспечивает построение (собственно визуализацию) трехмерной сцены. Параллельный сервер визуализации может использовать ресурсы нескольких вычислительных кластеров. Проведенные вычислительные эксперименты показали работоспособность и эффективность указанного подхода, применимого для обработки распределенных в глобальной сети сеточных данных.

Одним из наиболее мощных и наглядных методов визуализации трехмерных скалярных данных является визуализация изоповерхностей. Серверная часть алгоритма распределенной визуализации изоповерхностей описывается следующей последовательностью повторяющихся действий:

1) получение от клиентской части имен визуализируемых файлов;
2) чтение заголовков указанных файлов, определение диапазонов изменения пространственных координат и значения сеточной функции и передача соответствующих данных пользователю на клиентскую часть;
3) получение от клиентской части значений границ визуализируемого фрагмента изучаемой области и значения функции, соответствующей требуемой изоповерхности;
4) конструирование изоповерхности — построение описывающей изоповерхность триангуляции;
5) сжатие полученной триангуляции;
6) передача сжатых данных клиентской части.

Не будет преувеличением утверждение, что работоспособность системы распределенной визуализации трехмерных скалярных полей определяется главным образом качеством алгоритмов сжатия неструктурированных сеток. Именно неструктурированных сеток, поскольку несложно видеть, что даже при сечении плоскостью простейшей трехмерной кубической решетки образуется триангулированная поверхность (рис. 11). Появление триангуляции вызвано наличием разбиения каждого из кубов сетки на набор пирамид (например на шесть, как на рис. 11), обеспечивающих в большинстве случаев однозначность построения изоповерхности. Число описывающих изоповерхность узлов велико и по порядку величины может совпадать с числом узлов исходной трехмерной сетки, поэтому и необходим этап сжатия изоповерхности.

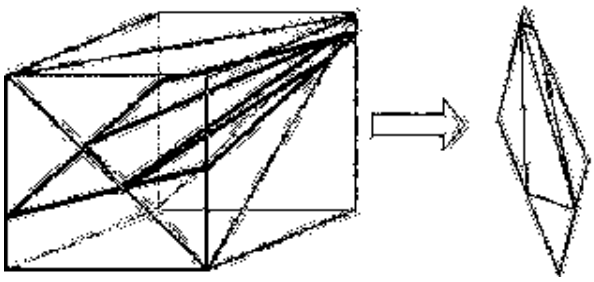


Рис. 11. Фрагмент пересекающей куб изоповерхности

Обсуждаемые алгоритмы предназначены для сжатия данных до размеров, допускающих их передачу через медленные каналы связи за короткое время, поэтому все они являются алгоритмами, сжимающими данные с потерей точности, часто значительной. Следующие положения частично служат оправданием такого подхода:

- число узлов, описывающих триангулированную изоповерхность, значительно превышает число пикселей стандартного монитора пользователя, так что при выводе часть информации будет неизбежно утеряна. При численном моделировании широко используются сгущающиеся сетки, причем разница между размерами самого длинного и самого короткого ребер может составлять несколько порядков. В результате большое число узлов сетки, описывающих зону ее сгущения, проецируется в малое число пикселей экрана — визуально они сливаются, значительно искажая изображение;
- визуально воспринимаются (и то не вполне адекватно) лишь основные контуры и формы трехмерного объекта, спроецированного на двумерный экран, — детали изображения воспринимаются хуже. Таким образом, при изучении объекта "в целом" деталями можно пожертвовать. При необходимости фрагмент объекта можно рассмотреть с большим увеличением и соответственно меньшей потерей точности.

Простейшие алгоритмы сжатия, основанные на предварительном уменьшении точности представления вещественных чисел, описывающих сетку, и последующей компрессии с помощью стандартных методов группового кодирования, кодирования Хаффмана, алгоритмов RLE, LZW [10—12], значительного выигрыша не дают. Основная причина в том, что большую часть объема данных о триангуляции составляет целочисленная информация, описывающая ее топологию — связи между узлами. Стандартными методами эта информация практически не сжимается. Существующие специальные методы, среди которых наиболее эффективен метод шелушения [13], существенно ориентированы на планарность графов и не обобщаются непосредственно на случай триангуляций, расположенных в трехмерном пространстве. Таким образом, основной интерес представляют алгоритмы, которые генерируют некоторую новую поверхность, аппроксимирующую исходную изоповерхность, но содержащую значительно меньшее количество узлов.

Для сжатия изоповерхностей можно использовать ряд алгоритмов, обладающих одним из следующих свойств:

- выполнением сжатия исходной изоповерхности с заданной точностью;
- обеспечением заданного объема данных, описывающих изоповерхность.

Рассматриваемые далее алгоритмы сжатия триангулированных поверхностей можно условно разделить на методы сжатия синтезом (или методы синтеза) и методы сжатия редукцией (методы редукции).

Методы синтеза относятся к алгоритмам, не гарантирующим точность восстановления изоповерхности. Их основные преимущества в том, что они, во-первых, обеспечивают сжатие данных до заданного размера и, во-вторых, позволяют значительно сократить объем информации о топологии аппроксимирующей поверхности [14, 9], передаваемой между серверной и клиентской частями системы визуализации.

Рассмотрим сжатие изоповерхности F, однозначно проецируемой на плоскость (x, y) (рис. 12,*а*). Описывающие изоповерхность точки могут быть разделены на внутренние (соседние к которым образуют простой цикл) и внешние. Удалив все внутренние точки (рис. 12,*б*) и часть внешних, получим ряд опорных точек аппроксимирующей поверхности G (рис. 12,*в*). Теперь построим триангуляцию внутренней области, последовательно добавляя вершины внутрь области. При определении координат очередной вершины будем использовать только информацию о координатах (x, y, z) опорных и уже добавленных вершин, где z — координата пересечения нормали к плоскости (x, y) с изоповерхностью $F(x, y)$.

Таким образом, для восстановления поверхности G необходимо передать на клиентскую часть только информацию (x, y, z) об опорных точках и значения z в добавленных точках. Нет необходимости передавать координаты (x, y) добавленных точек и топологию связей между ними, поскольку их можно полностью определить при восстановлении поверхности, просто повторив действия, выполненные при ее сжатии.

Пример полученной таким методом поверхности приведен на рис. 12,*г*. При ее построении был использован простой алгоритм добавления каждой из вершин (рис. 13):

1) выбор самого длинного ребра AC в уже построенной части поверхности G;
2) добавление точки E в середину выбранного ребра AC;

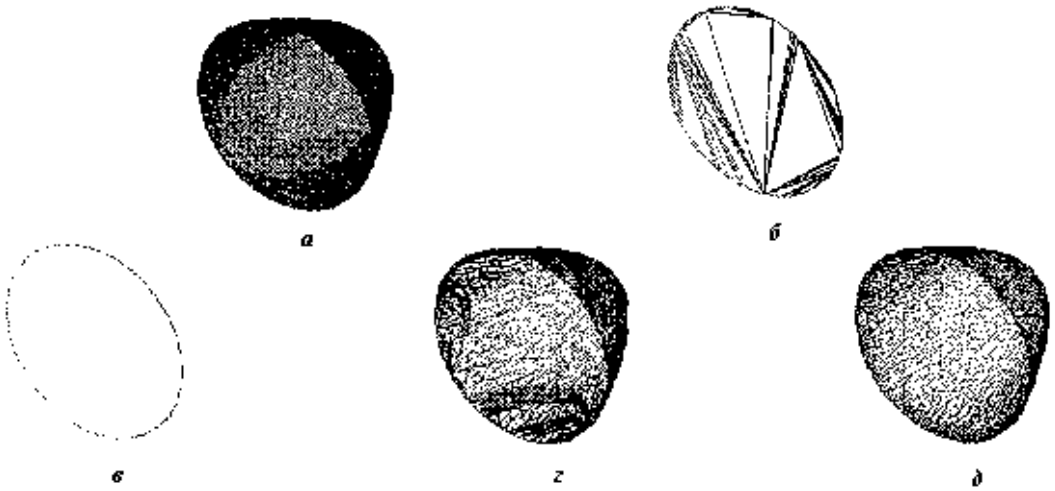


Рис. 12. Этапы сжатия изоповерхности: *а* — сжимамая изоповерхность; *б* — результат удаления внутренних точек; *в* — опорные точки аппроксимирующей поверхности; *г* — восстановленная поверхность; *д* — поверхность, восстановленная усовершенствованным методом

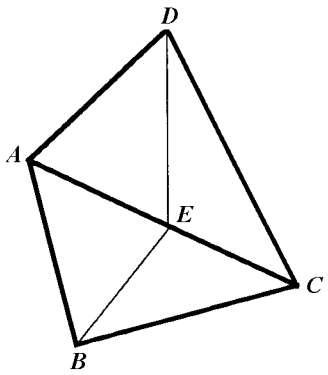


Рис. 13. Добавление в триангуляцию новой вершины

3) соединение добавленной точки E ребрами с противолежащими вершинами B, D треугольников, опирающихся на ребро AC;
4) определение уровня z изоповерхности в точке E.

Очевидно, что данный алгоритм позволяет воспроизвести изоповерхность на клиентской части системы. Однако простота алгоритма приводит к тому, что в узкие длинные треугольники, построенные на опорных точках, добавляется неоправданно много близких между собой точек (темные полосы на рис. 12,*г*). Использование более развитых алгоритмов позволяет получать вполне приемлемые результаты ценой незначительного увеличения времени формирования поверхности G (рис. 12,*д*). Хорошие результаты дает добавление точки в центр тяжести треугольника максимальной площади с последующей корректировкой триангуляции в соответствии с критерием Делоне [15].

Изложенный подход позволяет, во-первых, значительно уменьшить число точек (как правило, достаточно нескольких сотен точек для передачи формы поверхности); во-вторых, дополнительно снизить объем передаваемых данных за счет отсутствия необходимости передачи информации о топологии поверхности G.

Для аппроксимации поверхностей, которые нельзя однозначно спроецировать ни на какую плоскость (рис. 14,*а*), необходимо предварительно выполнить декомпозицию поверхности на связные фрагменты, каждый из которых может быть однозначно спроецирован на некоторую плоскость (рис. 14,*б*), в общем случае не параллельную никакой из координатных плоскостей. На рис. 14,*в* представлен результат сжатия методом синтеза тестовой поверхности — сферы с вырезанными отверстиями.

Алгоритмы *сжатия редукцией* не предполагают предварительного разбиения поверхности на части. Их основная идея заключается в удалении из исходной поверхности некоторого количества узлов таким образом, чтобы триангуляция, определенная на оставшихся точках, аппроксимировала исходную поверхность с требуемой точностью.

Редукция исходной поверхности выполняется с помощью нескольких шагов просмотра [16—18]. На каждом шаге исключение узлов происходит последовательно. Для каждого узла поверхности проверяется возможность его удаления без нарушения заданной точности аппроксимации. При удалении узла и опирающихся на него треугольников соседние с ним вершины соединяются друг с другом так, чтобы образовались треугольники, аппроксимирующие поверхность наилучшим образом. Для обеспечения возможности контроля точности аппроксимации на последующих шагах сохраняется некоторая информация об удаленных узлах и треугольниках. В простейшем случае удаленные узлы накапливаются в списках, ассоциированных с порожденными треугольниками. Для повышения качества аппрокси-

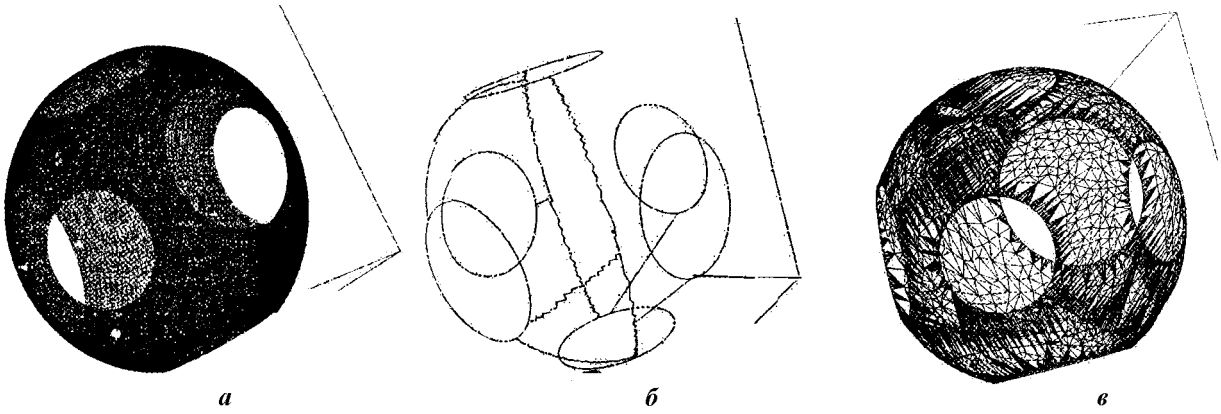


Рис. 14. Сжатие сферы с вырезанными отверстиями методом синтеза: *а* — триангулированная сфера с вырезанными отверстиями: 98 880 точек, 195 884 треугольников; *б* — линии, разделяющие однозначно проецируемые зоны: 3 483 точек; *в* — результат сжатия: 4 481 точек, 5 948 треугольников

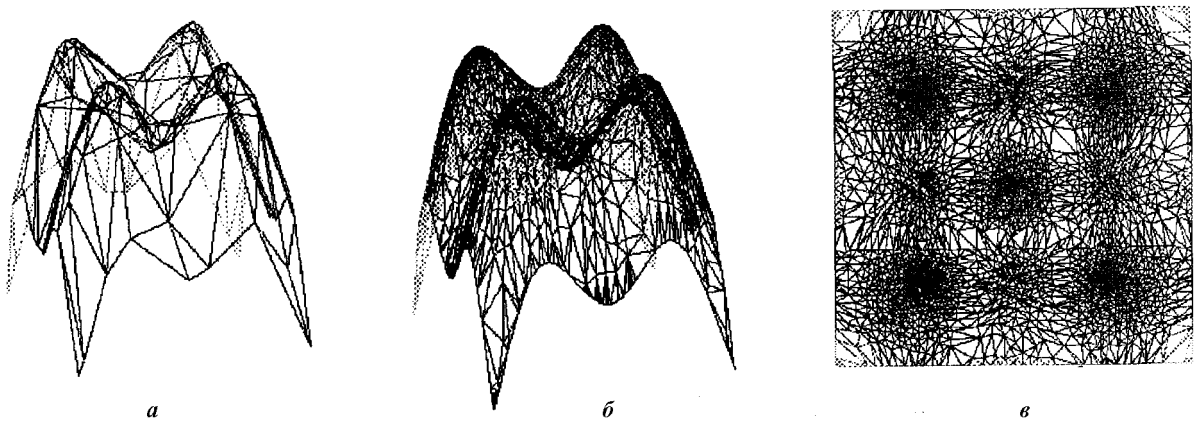


Рис. 15. Сжатие редукцией тестовой поверхности: *а* — с точностью 2%; *б* — с точностью 0,1%; *в* — на трех процессорах с точностью 0,1%

мации удаление узлов происходит итерационно, при этом на каждом шаге узлы удаляются равномерно по всей поверхности. Не удаляются узлы, соседние с ранее удаленными на этом шаге. Просмотр повторяется, если при выполнении последнего шага была удалена хотя бы одна точка.

Платой за соблюдение точности аппроксимации и высокое качество получаемых с помощью методов редукции визуальных образов является значительно большее, чем у методов синтеза, время сжатия. На рис. 15,*а,б* представлены результаты сжатия тестовой поверхности $f = x \sin x + y \sin y$, заданной в области $x \in [-4, 4]$, $y \in [-4, 4]$. На рис. 16 представлен результат сжатия с помощью редукции сферы, представленной на рис. 14,*а*.

Сжимаемая изоповерхность может иметь достаточно сложную топологию, например, на одно ребро может опираться более двух треугольников. Построить алгоритм обработки таких поверхностей с помощью методов синтеза достаточно сложно, но есть возможность за короткое время распознать наличие самопересечения и использовать для их обработки алгоритмы редукции. Пример самопересекающейся поверхности (круг пересекается фрагментом цилиндра) и результат ее сжатия с помощью метода редукции приведен на рис. 17.

Параллельный алгоритм построения и сжатия изоповерхности можно разработать на основе метода геометрического параллелизма [19]. В соответствии с ним каждый вычислительный модуль многопроцессорной системы обрабатывает компактную часть сетки (домен, полученный с помощью изложенных ранее методов). Передач данных между процессорами на этом этапе не требуется, что тем не менее не обя-

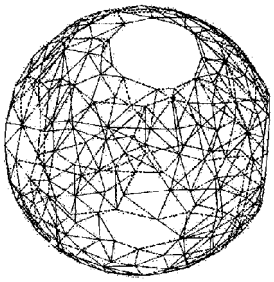


Рис. 16. Результат сжатия сферы с отверстиями с помощью редукции: 215 точек, 375 треугольников

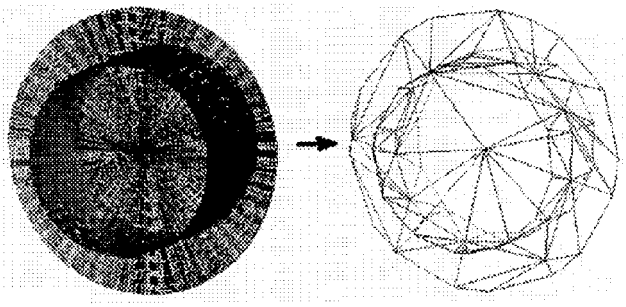


Рис. 17. Сжатие редукцией самопересекающейся поверхности (плоскость, пересекающая цилиндр) с точностью 5%

зательно обеспечивает высокую эффективность параллельной обработки.

Следует подчеркнуть, что при параллельной визуализации данных, определенных на сетках большого размера, в первую очередь ставится задача обеспечения самой возможности выполнения визуализации, а уже затем — снижения времени обработки для обеспечения возможности работы в интерактивном режиме. При этих условиях некоторые из процессоров могут большую часть времени не принимать участия в обработке, снижая таким образом эффективность использования вычислительной мощности системы в целом. Но это не является сколь-нибудь существенным недостатком хотя бы потому, что при интерактивной работе между запросами пользователя может проходить заметное время, в течение которого все процессоры простаивают.

После сжатия отдельными процессорами сформированных фрагментов поверхности результаты собираются на одном процессоре. На нем происходит объединение фрагментов в единое целое, "сшивка" границ фрагментов и сжатие уже всей поверхности. Полученная триангуляция передается по каналам Интернета или локальной сети на персональный компьютер пользователя, где и происходит окончательное построение изображения. Двухэтапное сжатие — сначала по частям, с сохранением границ доменов, затем всей поверхности, а фактически, именно границ между доменами — приводит к тому, что области, прилегающие к границам доменов, сжимаются хуже, чем внутренние зоны, и заметно выделяются на рисунке или на экране монитора. Пример сжатия тестовой поверхности (рис. 15,*б*) на трех процессорах приведен на рис. 15,*в*, на котором выделяются две горизонтальные полосы, соответствующие границам между тремя доменами.

Заключение

Рассмотренные в работе алгоритмы и методы распределенной обработки на многопроцессорных системах сеточных данных позволяют выполнять численное моделирование широкого круга задач и проводить интерактивную визуализацию результатов, полученных на неструктурированных многомерных сетках большого размера.

Автор выражает благодарность сотруднику ИВМ РАН Ю. В. Василевскому за помощь при подготовке трехмерной сетки, а также сотрудникам и аспирантам ИММ РАН и МФТИ П. С. Кринову, С. А. Сукову, С. В. Муравьеву и С. Н. Болдыреву за значительную помощь, оказанную при подготовке материала статьи, и вклад, внесенный в разработку, реализацию и изучение представленных алгоритмов и методов.

Список литературы

1. *Hendrickson B., Leland R.* Multilevel algorithm for partitioning graphs // Proc. of "Supercomputing'95". San Diego, CA, 1995.
2. *Hendrickson B., Leland R.* An improved spectral graph partitioning algorithm for mapping parallel computations // SIAM J. Sci. Comp. 1995. Vol. 16, No 2. P. 452—469.
3. *Fiedler M.* Eigenvectors of acyclic matrices // Czechoslovak Mathematical Journal. 1975. Vol. 25(100). P. 607—618.
4. *Fiedler M.* A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory // Ibid. P. 619—633.
5. *Karypis G., Kumar V.* Multilevel graph partitioning schemes // ICPP. 1995. Vol. 3. P. 113—122.
6. *Fiduccia C., Mattheyses R.* A linear time heuristic for improving network partitions // Proc. 19th. IEEE Design Automation Conference. 1982. P. 175—181.
7. *Адельсон-Вельский Г. М., Ландис Е. М.* Один алгоритм организации информации // Докл. АН СССР. 1962. Т. 146, № 2. С. 263—266.
8. *Кнут Д.* Искусство программирования. Т. 3. Сортировка и поиск: Пер. с англ. М.: "Вильямс", 2001.
9. *Iakobovski M. V., Karasev D. E., Krinov P. S., Polyakov S. V.* Visualisation of grand challenge da-

ta on distributed systems // Mathematical Modeling. Problems, Methods, Applications. Proc. of the 4th Int. Mathematical Modeling Conference. June 27—July 1, 2000. Moscow, Russia — N.-Y.: Kliwer Academic / Plenum Publishers, 2001. P. 71—78.

10. *Huffman D. A.* A method for the construction of minimum redundancy codes // Proc. of IRE. 1952. Vol. 40. P. 1098—1101.
11. *Климов А. С.* Форматы графических файлов. К.: НИПФ "ДиаСофт Лтд", 1995.
12. *Ватолин Д., Ратушняк А., Смирнов М., Юкин В.* Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. М.: ДИАЛОГ-МИФИ, 2002.
13. *Скворцов А. В., Костюк Ю. Л.* Сжатие координат узлов триангуляции // Изв. вузов. Физика. 2002. № 5. С. 26—30.
14. *Iakobovski M. V., Krinov P. S.* Visualisation of grand challenge data on distributed multiprocessor systems // 5th Int. Congress of Mathematical Modeling. Books of abstracts. Vol. 1. M.: JANUS-K, 2002.
15. Эксперимент на дисплее: Первые шаги вычислительной физики / Под ред. Р. З. Сагдеева. М.: Наука, 1989.
16. *Кринов П. С., Якобовский М. В., Муравьев С. В.* Сжатие и визуализация триангулированных поверхностей // V науч. конф. МГТУ "Станкин" и "Уч.-науч. центра мат. моделирования МГТУ «Станкин» — ИММ РАН". М.: Янус-К, ИЦ МГТУ "Станкин", 2003. С. 32—42.
17. *Кринов П. С., Якобовский М. В., Муравьев С. В.* Визуализация данных большого объема в распределенных многопроцессорных системах // Мат. III Межд. науч.-практ. семинара / Под ред. Р. Г. Стронгина. Н. Новгород: Изд-во Нижегородского госуниверситета, 2003. С. 81—88.
18. *Krinov P. S., Iakobovski M. V., Muravyov S. V.* Large data volume visualization on distributed multiprocessor systems // Proc. of Int. Conf. "Parallel Computational Fluid Dynamics". Book of Abstracts. M.: JANUS-K, 2003. P. 301—304.
19. Транспьютеры. Архитектура и программное обеспечение: Пер. с англ. / Под ред. Г. Харпа. М.: Радио и связь, 1993.

Статья поступила в редакцию 03.03.04.