УДК 519.688

ОБЩИЙ ПОДХОД К РЕАЛИЗАЦИИ МЕТОДОВ ПОСТРОЕНИЯ ТРИАНГУЛЯЦИЙ НЕЯВНО ЗАДАННЫХ ПОВЕРХНОСТЕЙ, ИСПОЛЬЗУЮЩИХ РАЗБИЕНИЕ ПРОСТРАНСТВА НА ЯЧЕЙКИ

А. Ю. Дижевский[1]

Описаны наиболее популярные алгоритмы триангуляции трехмерных объектов, разбивающие пространство на кубические и тетраэдрические ячейки. В работе предлагается общий подход к построению триангуляции трехмерных объектов, использующий разбиение пространства на произвольные ячейки. Представлена реализация данного подхода на примере новых методов триангуляции, разбивающих пространство на пирамиды и призмы. Приведены особенности реализации всех описанных методов. Выполнен сравнительный анализ качества получаемых триангуляций.

1. Введение. Необходимость в построении триангуляции поверхности уровня для функции, заданной в некоторой области трехмерного пространства, и последующей визуализации возникает во многих областях исследований: в физике, математике, геологии, медицине и др. Поверхность уровня задается формулой $f(x, y, z) - c = 0$, где c — пороговая константа. В геологии для нахождения, визуализации и определения топологии подземных нефтяных бассейнов используются сейсмические данные о пористости [11]. В медицине обработка и визуализация данных, полученных с томографа, очень важна для диагностики и планирования лечения. При этом наряду с двухмерной визуализацией часто бывает полезно рассмотреть трехмерное изображение изучаемого объекта (раковые опухоли, особенности сосудов, отложение кальция в сосудах, камни в почках и т.п.). Здесь функция плотности в пространстве $f(x, y, z)$ строится по серии полученных на томографе параллельных сечений, каждое из которых представляет собой прямоугольную матрицу, содержащую плотности исследуемого объекта в точках сечения. В промежутках между сечениями применяется интерполяция.

Процесс визуализации трехмерных объектов в современных системах компьютерной графики проходит через ряд этапов:

— получение исходной триангуляции поверхности;

— ее упрощение (simplification) при необходимости (см. [14]);

— переупорядочивание треугольников с целью образования полос (triangle strips) и вееров (fans), ускоряющих процесс передачи данных в видеокарту [13];

— построение прогрессивных мешей (progressive meshes) для обеспечения разных уровней детализации и компактного хранения данных [12];

— сжатие данных;

— визуализация.

В настоящей статье рассматривается лишь самый первый этап в этой цепочке. Обсуждаются алгоритмы, строящие триангуляцию поверхности за один проход без последующих корректировок полученных треугольников и без учета топологических несвязностей [9]. Полученный набор треугольников может визуализироваться стандартными графическими пакетами, такими как OpenGL или Directx.

Алгоритм “марширующие кубы”, предложенный Лоренсеном [1], является самым ранним способом получения поверхностей уровня функций в трехмерном пространстве. Данный алгоритм разбивает пространство на кубические ячейки и выполняет аппроксимацию поверхности внутри каждой ячейки.

Методы из класса “марширующие тетраэдры” строят тетраэдрические сети и аппроксимируют исходную поверхность внутри каждого тетраэдра. Обычно при построении тетраэдрических сетей используется разбиение пространства на кубы, а затем каждый куб разбивается на тетраэдры.

Пространство можно разбивать на ячейки рекурсивно; область пространства (если она пересекается с исходной поверхностью) разбивается на восемь областей путем ее деления на две части вдоль каждой из трех осей [7]. Полученные области хранятся в древовидной структуре. Процесс деления начинается с исходной области (формы параллелепипеда) и останавливается, когда размер полученных областей меньше

[1] Московский государственный университет им. М. В. Ломоносова, механико-математический факультет, Ленинские горы, 119899, Москва; e-mail: mathlog@yandex.ru

заданного значения. Область применения таких алгоритмов ограничена; например, на медицинских данных они плохо работают из-за недостаточной гладкости функции плотности и наличия шумов. В данной работе такие алгоритмы не рассматриваются.

В связи с тем, что алгоритмическая реализация методов триангуляции производится коммерческими компаниями, детали реализации, как правило, закрыты. Целью данной работы является подробное описание программной реализации алгоритмов "марширующие кубы", "марширующие тетраэдры 5 и 6", алгоритма Скалы, а также разработка новых алгоритмов триангуляции на основе предложенного общего подхода к построению триангуляций.

Общий подход использует разбиение пространства на произвольные фигуры. Реализация этого подхода иллюстрируется нами на примере алгоритмов, разбивающих пространство на пятивершинные пирамиды и шестивершинные призмы. Методы триангуляции с помощью разбиения пространства на фигуры, отличные от куба и тетраэдра, не были представлены ранее в литературе. Показано, что основная сложность алгоритмов триангуляции заключается в построении сети фигур и разработке таблиц пересечения фигур и поверхности. Разработанные алгоритмы для двух типов фигур проще в реализации, поскольку они используют меньшие таблицы, чем "марширующие кубы", и более простые сети, чем алгоритмы из семейства "марширующие тетраэдры".

2. Постановка задачи триангуляции. Поставим задачу триангуляции трехмерной поверхности, заданной в области пространства формулой $f(x, y, z) = c$, где c — заданный уровень. Требуется аппроксимировать данную поверхность треугольниками [10]. Для простоты реализации методов триангуляции будем считать, что поверхность задана в кубе с начальной точкой (x_0, y_0, z_0), ребром 1 и уровнем $c = 0$.

3. Метод "марширующие кубы". Алгоритм "марширующие кубы" можно разбить на два этапа:
— разбиение области пространства на прямоугольные параллелепипеды;
— построение триангуляции области поверхности, пересекающей параллелепипед.

3.1. Разбиение области пространства на прямоугольные параллелепипеды и особенности реализации. Для простоты изложения будем рассматривать векторное пространство.

Область пространства — куб, который разбивается на равные n частей вдоль осей x, y, z. В дальнейшем куб будем называть исходным кубом, а элементы (кубы) разбиения — кубами разбиения. В результате получаем n^3 кубов разбиения.

При пересечении куба и исходной поверхности образуются треугольники. Вершины треугольников лежат на ребрах куба и вычисляются с помощью линейной интерполяции по значениям функции на вершинах куба. При программной реализации возникает необходимость хранения номеров вершин треугольников. Будем хранить их в массиве TriangleVertices. Для программной реализации также следует занумеровать ребра кубов разбиения и поместить их в массив EdgeArray. Пример нумерации ребер для $n = 2$ приведен на рис. 1.

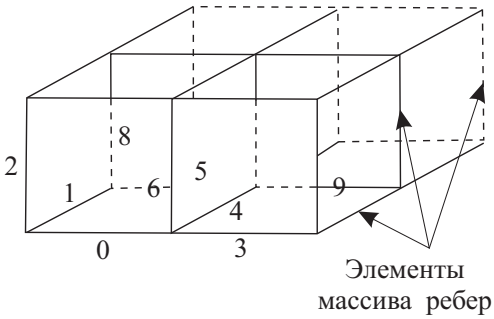

Рис. 1. Разбиение куба и нумерация ребер

Данную нумерацию можно представить следующим образом:

$$\text{номер ребра} = \text{номер узла}\times 3 + \text{подындекс (от 0 до 2)}. \tag{1}$$

Здесь узел — левая нижняя вершина каждого из кубов разбиения. Следовательно, количество узлов равно n^3. Для правой, задней и верхней граней исходного куба нужно увеличить количество узлов на один в каждом из трех направлений x, y, z, т.е. количество узлов будет $(n+1)^3$. Элемент массива EdgeArray — три ссылки на ребра (два горизонтальных и одно вертикальное), номера показаны на рис. 1. Для правой, задней и верхней граней исходного куба будут использоваться не все индексы. Например, для правой грани подындекс (см. формулу (1) для номеров ребер) будет равен 1 или 2, так как подындекс с номером 0 уже находится вне исходного куба. На псевдокоде можно определить функцию для получения и для установки индекса ребра по заданному узлу (его координатам x, y, z) и подындексу следующим образом:

```
GetTheEdgeNumber(x, y, z coordinates of the node, subindex)
(
   NodeNumber = (m_xMax*m_yMax)*z + (m_xMax)*y + x
   EdgeNumber =  EdgeArray[NodeNumber*3 + subindex]
   return EdgeNumber;//искомый индекс ребра
)
```

Здесь $0 \leqslant$ x $\leqslant$ m_xMax, $0 \leqslant$ y $\leqslant$ m_yMax, $0 \leqslant$ z $\leqslant$ m_zMax и m_xMax, m_yMax, m_zMax — максимальные номера узла для каждого из трех направлений. В данном случае эти индексы будут равны $(n + 1)$.

Треугольники триангуляции будем хранить в массиве Triangles. Элементом массива являются три ссылки на вершины треугольника, т.е. три номера из массива вершин. В свою очередь, элементами массива вершин треугольников TriangleVertices будут вершины треугольников и нормали к вершинам. Заполнение массива вершин описано ниже. Нормали к вершинам используются при визуализации для создания эффекта гладкости поверхности. Нормали задаются градиентом триангулируемой поверхности. Триангуляция — это массив треугольников Triangles и массив вершин треугольников TriangleVertices.

При заполнении массива вершин понадобится функция, которая находит точку пересечения поверхности и отрезка. Если пересечение не пустое, то функция добавляет точку пересечения в массив вершин триангуляции. Отрезок задается векторами v0 и v1. На псевдокоде эта операция реализуется следующим образом:

```
DefineIntersectionPoint(v0, v1, value0, value1, normal0, normal1)
(
    If (value0 <= 0 and value1 <= 0) or (value0 > 0 and value1 > 0)
       The empty intersection;

    coeff1 = value0 / (value0 - value1);
    coeff0 = 1 - coeff1;
    v = v0*coeff0+v1*coeff1;
    bNormal  = normal0*coeff0 + normal1*coeff1;
    The vector v and the normal bNormal
    are added to the array TriangleVertices.
)
```

Здесь value0 и value1 — значения исходной функции на концах отрезка, normal0 и normal1 — нормали в точках v0, v1 к поверхности.

Точку пересечения можно задать иначе, например так, чтобы точка пересечения была серединой ребра (v0, v1), как это было предложено в [5].

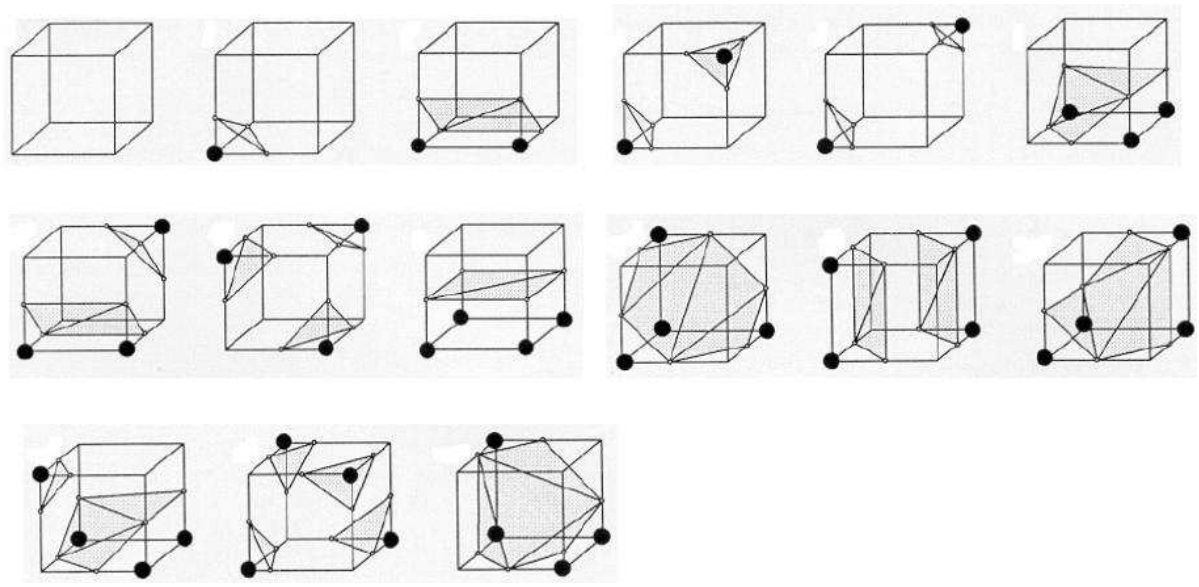

Рис. 2. Варианты пересечения куба и поверхности

3.2. Триангуляции области поверхности, пересекающей параллелепипед. На вершинах куба функция $f(x, y, z)$ может принимать положительные и отрицательные значения. Если на ребре куба разбиения функция принимает различные знаки, то поверхность, заданная этой функцией, пересекает это ребро.

Пусть имеется 8-битовая строка, соответствующая знакам функции на вершинах куба (их восемь). Если на вершине куба функция принимает отрицательное значение, то значение бита строки равно 0,

иначе это значение равно 1. Тогда количество разных вариантов пересечения равно $2^8 = 256$. Используя симметрию и вращение, все 256 способов можно свести к 15 (рис. 2), как это сделано в [1].

Рассмотрим реализацию этих случаев. Обозначения индексов вершин и ребер показаны на рис. 3.

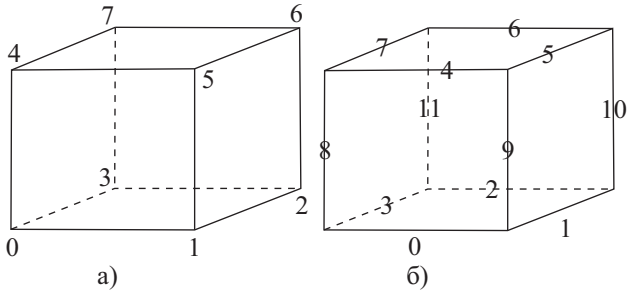

Рис. 3. Индексы вершин (а) и ребер (б) для алгоритма “марширующие кубы”

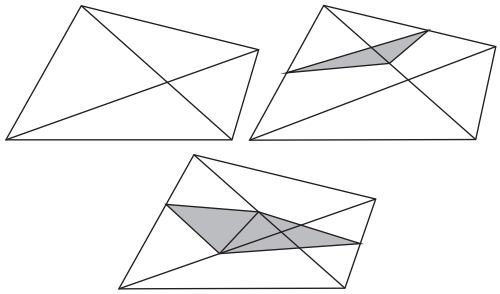

Рис. 4. Варианты пересечения тетраэдра и поверхности

Например, если значение на вершине 3 отрицательное, а на других вершинах положительное, то пересечением будет треугольник на ребрах 2, 3, 11. При реализации алгоритма использована таблица, содержащая варианты пересечения для каждого из 256 случаев. По 8-битовой строке таблица выдает номера ребер пересечения. Получение этой строки реализовано следующим образом:

```
cubeindex = 00000000;
если (val[0] < 0), устанавливаем 0-ю цифру справа cubeindex в 1;
если (val[1] < 0), устанавливаем 1-ю цифру справа cubeindex в 1;
если (val[2] < 0), устанавливаем 2-ю цифру справа cubeindex в 1;
если (val[3] < 0), устанавливаем 3-ю цифру справа cubeindex в 1;
если (val[4] < 0), устанавливаем 4-ю цифру справа cubeindex в 1;
если (val[5] < 0), устанавливаем 5-ю цифру справа cubeindex в 1;
если (val[6] < 0), устанавливаем 6-ю цифру справа cubeindex в 1;
если (val[7] < 0), устанавливаем 7-ю цифру справа cubeindex в 1.
```

Здесь val$[i]$ — значение функции на i-й вершине куба. Назовем эту функцию инициализацией cubeindex. Ниже представлена таблица с первыми двенадцатью из 256 случаев пересечения куба и поверхности; полную таблицу можно найти в [6]. Например, для случая, когда поверхность на третьей вершине положительна, имеем cubeindex = 00001000 или 8. По таблице TriangTable получаем номера пересекаемых ребер (3, 2, 11).

```
TriangTable[256][16] =
((-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(0, 1, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(1, 8, 3, 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(0, 8, 3, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(9, 2, 10, 0, 2, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(2, 8, 3, 2, 10, 8, 10, 9, 8, -1, -1, -1, -1, -1, -1, -1),
(3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(0, 11, 2, 8, 11, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(1, 9, 0, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1),
(1, 11, 2, 1, 9, 11, 9, 8, 11, -1, -1, -1, -1, -1, -1, -1),...
)
```

4. Алгоритмы класса “марширующие тетраэдры”. Алгоритмы из этого класса разбивают пространство на тетраэдры. Внутри каждого тетраэдра строится триангуляция. Рассмотрим возможные случаи пересечения тетраэдра и поверхности.

4.1. Триангуляция внутри тетраэдра. Возможны 16 случаев пересечения тетраэдра и поверхности, поскольку у тетраэдра четыре вершины (т.е. существует 2^4 вариантов). С помощью поворотов эти случаи можно свести к трем. На рис. 4 изображены три неэквивалентных случая.

Введем 4-битовую строку, определяющую вариант пересечения поверхности и тетраэдра [8]. Пусть бит из строки установлен в 0, если функция принимает отрицательное значение на вершине тетраэдра, и 1, если положительное. Теперь занумеруем ребра и вершины тетраэдра.

Получить 4-битовую строку cubeindex можно следующим образом:

```
cubeindex = 0;
если (val[0] < 0), устанавливаем 0-ю цифру справа cubeindex в 1;
если (val[1] < 0), устанавливаем 1-ю цифру справа cubeindex в 1;
если (val[2] < 0), устанавливаем 2-ю цифру справа cubeindex в 1;
если (val[3] < 0), устанавливаем 3-ю цифру справа cubeindex в 1.
```

Здесь $\mathrm{val}[i]$ — значение поверхности на i-й вершине. В [6] представлена таблица, которая 4-битовой строке ставит в соответствие набор пересекаемых ребер (треугольник, два треугольника или пустое множество).

На рис. 5 представлена нумерация ребер и вершин тетраэдра. Данная нумерация используется в таблице случаев пересечения тетраэдра и поверхности. Получение номера строки таблицы можно производить как и в методе "марширующие кубы" (получение строки cubeindex длиной в четыре бита).

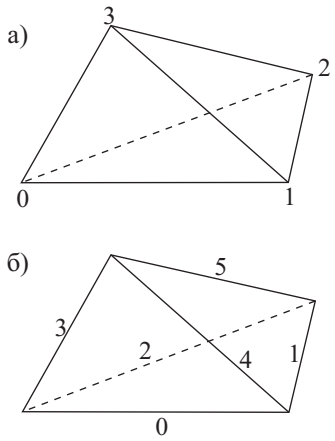

Рис. 5. Нумерация вершин (а) и ребер (б) тетраэдра

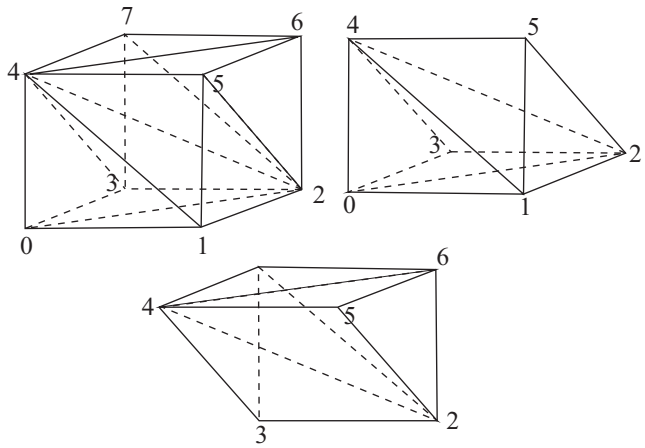

Рис. 6. Разбиение куба на шесть тетраэдров

Например, если значение функции на вершине 1 положительно, а на других отрицательно, то получим строку 0010 = 2. При помощи таблицы случаев для тетраэдра определяется треугольник, проходящий через ребра 0, 1, 4.

4.2. Алгоритм "марширующие тетраэдры 6". Алгоритм "марширующие тетраэдры 6" ("MT6") был предложен Гуезеком [2]. Пространство разбивается на кубические ячейки. Каждая ячейка разбивается на шесть тетраэдров (рис. 6).

Получаем тетраэдры:

```
1) 0 1 2 4; 2) 0 2 3 4; 3) 3 2 4 1; 4) 2 4 5 6; 5) 2 6 7 4; 6) 1 2 5 4.
```

Нумерация ребер и вершин куба представлена на рис. 7. Номера ребер с 0 по 11 совпадают с номерами ребер для алгоритма "марширующие кубы" (см. п. 3.2).

4.2.1. Разбиение области пространства на тетраэдры и особенности реализации. Область пространства разбивается на кубические ячейки. В кубических ячейках проведены диагонали граней и диагональ ячейки (рис. 7).

Занумеруем ребра кубов с учетом дополнительно образованных диагоналей, двигаясь по слоям вдоль оси z снизу вверх. Для каждого слоя определим два типа массивов ребер (для "горизонтальных" и "вертикальных" ребер).

На рис. 8 показана нумерация "горизонтальных" ребер. Каждому узлу (левой нижней вершине квадрата) соответствует три ребра. Получение индекса ребра по координатам узла (x, y) и подындексу можно производить так же, как и для метода "марширующие кубы", без учета z-координаты узла (процесс идет на слое).

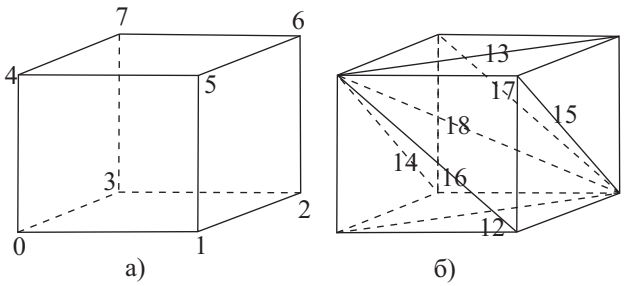

Рис. 7. Нумерация вершин (а) и ребер (б) куба для алгоритма “MT6”

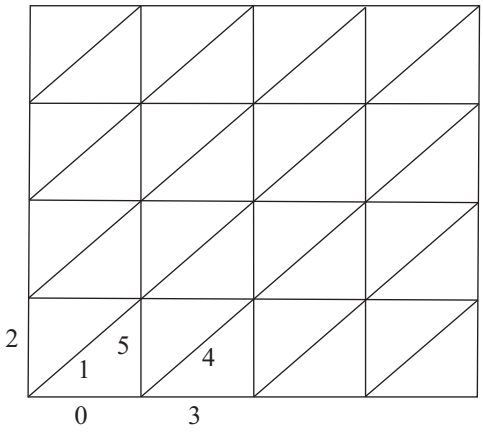

Рис. 8. Нумерация “горизонтальных” ребер для “MT6”

Для вертикальных ребер нумерация аналогична, только узлу соответствует четыре ребра. Для узла (дальней левой нижней вершины) на рис. 7 ребра имеют номера 14, 16, 18. Получение индекса ребра по заданному узлу и подындексу аналогично вышеописанному. Подиндекс принимает значения от 0 до 3.

4.3. Алгоритм “марширующие тетраэдры 5”. Алгоритм “марширующие тетраэдры 5” (“MT5”) был предложен Канейро [4]. Область пространства разбивается на кубические ячейки; каждая ячейка разбивается на пять тетраэдров (рис. 9).

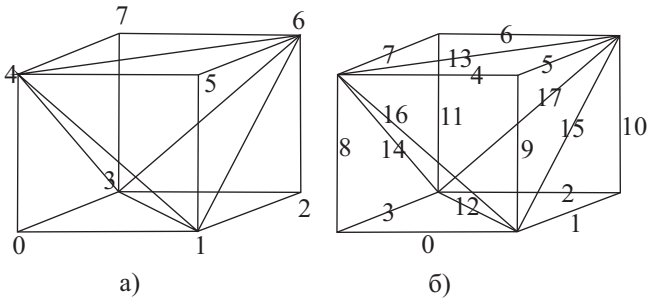

Рис. 9. Нумерация вершин (а) и ребер (б) куба для алгоритма “MT5”

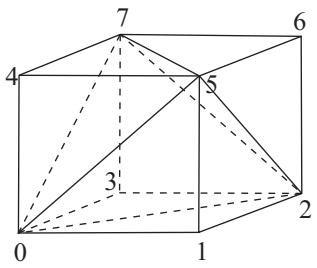

Рис. 10. Симметричное разбиение относительно центра ячейки

Получаем тетраэдры:

```
1) 0 1 3 4; 2) 1 2 3 6; 3) 4 5 6 1; 4) 4 6 7 3; 5) 1 3 6 4.
```

Кроме этого разбиения куба на тетраэдры необходимо использовать разбиение, симметричное относительно центра ячейки (на рис. 10). Симметричное разбиение необходимо использовать для заполнения пространства тетраэдрами. Кубические ячейки разбиваются на два типа тетраэдров (можно сказать, что получаются два типа ячеек). Ячейка одного типа имеет восемь соседей другого типа.

4.3.1. Разбиение области пространства на тетраэдры и особенности реализации. Пронумеруем ребра кубов с учетом диагоналей, двигаясь по слоям вдоль оси z снизу вверх (память выделяется только под один слой). Для каждого слоя будем хранить ребра в трех массивах. Ребра с верхних и нижних граней кубов слоя будем хранить в двух массивах одного типа (массив “горизонтальных” ребер). Остальные ребра будут храниться в массиве другого типа (массив “вертикальных” ребер).

На рис. 11 показана нумерация “горизонтальных” ребер. В данной нумерации каждому узлу (левой нижней вершине квадрата) соответствует три ребра. Получение индекса ребра по заданному узлу (его координатам x, y) и подындексу можно производить так же, как для метода “марширующие кубы”, без учета z-координаты узла. Горизонтальные ребра с подындексом 1 являются одной из диагоналей квадрата — в зависимости от номера узла в пространстве. В данном случае проверяется условие четности суммы $i + j + k$, где i, j, k — координаты узла вдоль осей x, y, z.

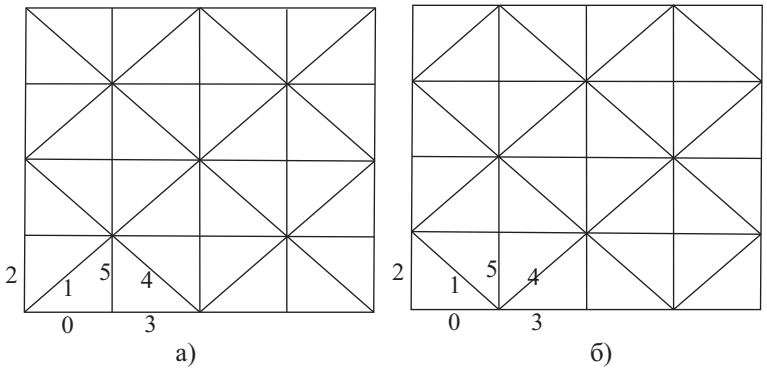

Рис. 11. Нумерация “горизонтальных” ребер для алгоритма “MT5”: а) нижние грани, б) верхние грани

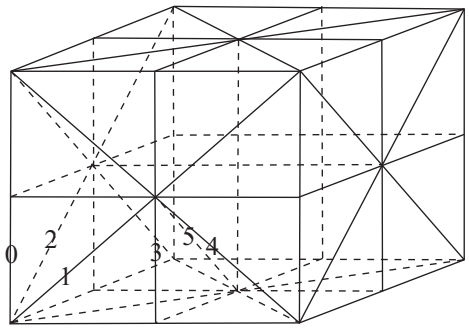

Рис. 12. Нумерация “вертикальных” ребер

Для “вертикальных” ребер каждому узлу будет соответствовать также три ребра. На рис. 12 показана нумерация “вертикальных” ребер. Здесь также проверяется условие четности номера узла, определенное ранее.

Для простоты реализации можно использовать шаблон, полученный из восьми ячеек. Единственное требование к разбиению на ячейки — четность количества элементов разбиения вдоль каждой из трех осей, так как шаблон является объединением соседних двух ячеек вдоль каждого направления.

4.4. Алгоритм Скалы. Пространство разбивается на кубические ячейки. Данный метод [3] заключается в том, что тетраэдры строятся не в ячейке, а на стыке двух ячеек. Для каждой ячейки добавляется дополнительная вершина — ее центр.

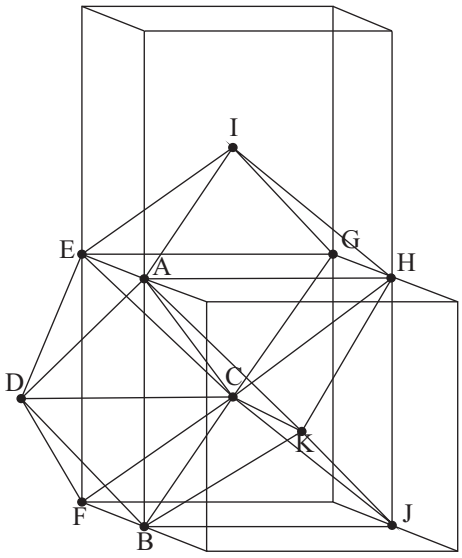

Рис. 13. Шаблон тетраэдров алгоритма Скалы

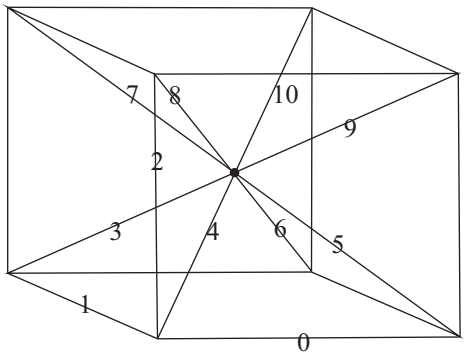

Рис. 14. Нумерация ребер для алгоритма Скалы

На рис. 13 показаны 12 тетраэдров (DEAC, DABC, DFBC, DFEC, IEAC, IAHC, IGHC, IEGC, KAHC, KHJC, KJBC, KABC), построенные данным способом.

4.4.1. Разбиение области пространства на тетраэдры и особенности реализации. Пронумеруем ребра кубов с учетом дополнительных диагоналей, двигаясь по слоям вдоль оси z снизу вверх (память выделяется под два соседних слоя). Для каждого слоя будем хранить ребра в массиве. Элемент массива — это ссылка на ребро. Количество элементов в массиве равно $11n^2$. Нумерация ребер представлена на рис. 14.

Таким образом, тетраэдрами будет заполнена внутренняя область исходного куба. Для заполнения куба с границей в массив вершин добавляются центры граней кубов разбиения, лежащих на границе исходного куба, поэтому к каждой грани куба разбиения на границе присоединятся по четыре дополнительных тетраэдра. Данный процесс можно представить также следующим способом: рассмотрим разбиение

на $(n+2)$ элемента вдоль каждой из осей, центры граничных кубов разбиения смещаем на половину ребра в сторону центра исходного куба и строим тетраэдрическую сеть, которая заполнит куб из n^3 элементов разбиения.

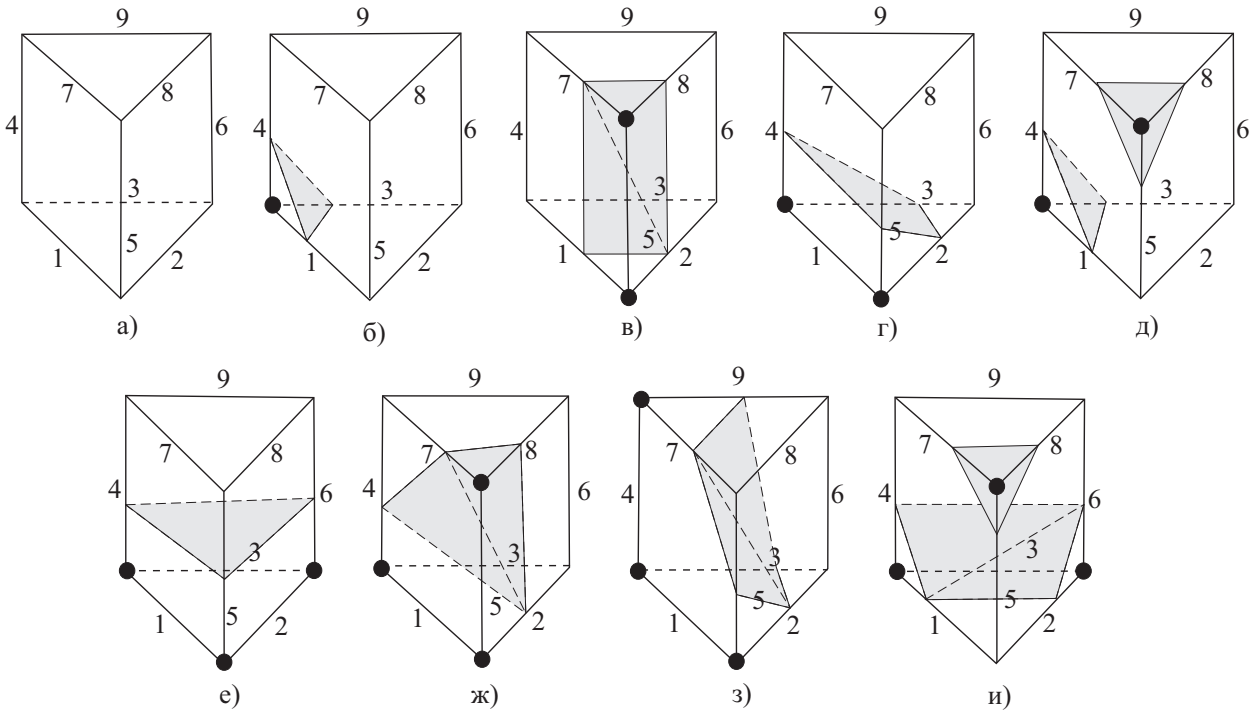

Рис. 15. Варианты пересечения призмы и поверхности

5. Алгоритм “Marching prisms”. Заполним пространство шестивершинными призмами, в основании которых лежат правильные треугольники.

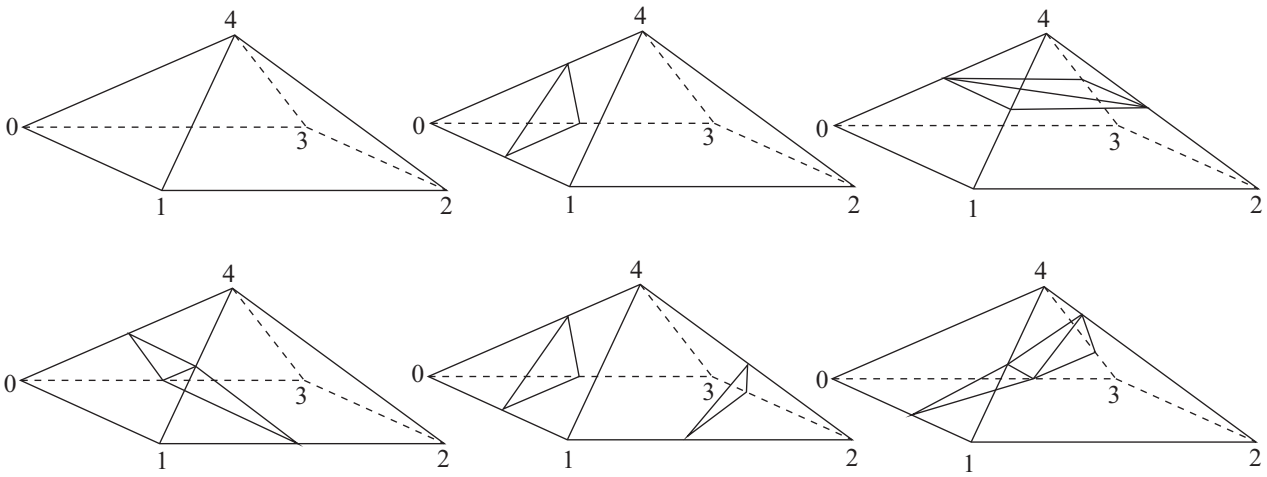

Рис. 16. Варианты пересечения пирамиды и поверхности

Возможны 64 случая пересечения призмы и поверхности, которые мы будем хранить в таблице из 64 строк. Каждый из 64 элементов таблицы состоит из 10 номеров вершин. Каждые три соседних номера обозначают треугольник. Если номер равен -1, то за ним нет треугольников. Максимально возможное количество треугольников для каждого случая пересечения равно трем, поэтому в строке таблицы имеется девять элементов плюс терминальный элемент. Такая структура хранения вариантов пересечения очень удобна для считывания индексов треугольников. Три значения из каждой строки таблицы обозначают стороны треугольника, полученного на пересечении поверхности и призмы. Если первое значение из тройки равно -1, то процесс считывания треугольников для строки таблицы останавливается.

Все 64 случая пересечения призмы с поверхностью можно свести к девяти при помощи поворотов

и симметрии. На рис. 15 показаны все девять случаев. Отмечены вершины, на которых функция имеет положительные значения.

Перечислим количество возможных случаев для каждого из девяти вариантов пересечения: а) 1, б) 6, в) 3, г) 6, д) 6, е) 2, ж) 6, з) 6, и) 1. Случаи, где количество положительных вершин больше трех, не рассмотрены, поскольку возможно обращение знака функции, т.е. можно рассмотреть для четырех положительных вершин случаи в), г), д), где знак плюс изменен на минус, и наоборот. Получение номера строки таблицы можно производить так же, как и в алгоритме "марширующие кубы" (получение строки cubeindex длиной в шесть бит).

6. Триангуляция пятивершинными пирамидами. Область пространства разбивается на кубические ячейки. Каждая ячейка разбивается на шесть пятивершинных пирамид. Вершинами пирамид являются центр ячейки и четыре вершины с каждой из шести граней куба. После этого строятся треугольники на пересечении пирамиды и поверхности.

Возможны 32 варианта пересечения пирамиды и поверхности, которые можно свести к шести неэквивалентным. На рис. 16 представлены шесть неэквивалентных вариантов. Отмечены вершины, на которых исходная функция принимает положительные значения.

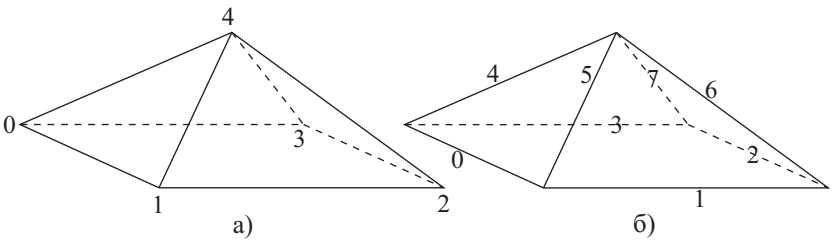

Рис. 17. Нумерация вершин (а) и ребер (б) пирамиды

Для программной реализации данного способа триангуляции используются нумерация ребер, принятая в методе Скалы, и таблица из 32 случаев пересечения пирамиды и поверхности. Каждая строка таблицы состоит из 10 индексов, так как максимально возможное количество треугольников на пересечении равно трем и каждый треугольник проходит через три ребра плюс терминальный индекс. Получение номера строки таблицы производится как в методе "марширующие кубы" (получение строки cubeindex длиной в пять бит). Нумерация ребер и вершин пирамиды для таблицы пересечения показана на рис. 17.

7. Результаты сравнительного анализа. Ниже сравниваются описанные методы по следующим параметрам:

— *aspect ratio*: характеризует качество получаемых треугольников;

— *deviation*: среднее отклонение от исходной поверхности;

— *deviation norm*: среднее отклонение нормалей треугольников от нормалей касательных плоскостей.

Параметр *aspect ratio* для треугольника вычисляется как отношение радиусов описанной и вписанной в треугольник окружностей. *Aspect ratio* триангуляции вычисляется как среднее арифметическое параметра *aspect ratio* для каждого треугольника из триангуляции.

Параметр *deviation* характеризует качество аппроксимации исходной поверхности полученной триангуляцией. Этот параметр вычисляется следующим образом. Для треугольника из триангуляции вычисляются центр пересечения медиан (центр масс) и вектор нормали к треугольнику. Определяется точка пересечения исходной поверхности и прямой, параллельной вектору нормали и проходящей через центр масс. Расстояние между полученной точкой пересечения и центром масс есть отклонение для треугольника. Для триангуляции данный параметр усредняется по всем треугольникам.

Параметр *deviation norm* характеризует меру правильности расположения треугольников по отношению к исходной поверхности. Этот параметр для треугольника вычисляется как синус угла между направляющим вектором касательной плоскости и вектором нормали к треугольнику. Для триангуляции этот параметр усредняется по всем треугольникам. Если параметр *deviation norm* для треугольника равен 0, то касательная плос-

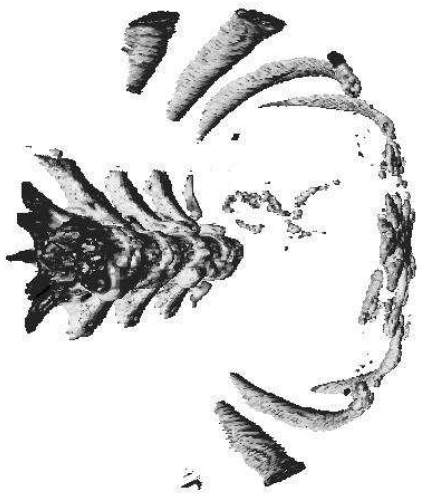

Рис. 18. Триангуляция томографических снимков грудной полости

кость параллельна плоскости треугольника, а если он равен 1, то перпендикулярна. *Deviation norm* близок к 1 для некоторых треугольников цилиндрических поверхностей — это схожая конструкция с "сапогом Шварца".

Входными данными являются набор томографических снимков и неявные поверхности, заданные формулой $f(x, y, z) = 0$.

Исходная область (куб с ребром 2 и центром в начале координат) делится на 200 частей вдоль каждой из трех осей.

На данных томографических снимков определена функция плотности от координат точки $p(x, y, z)$. Данная функция принимает значения от 0 до 5000 (приблизительно). Плотность воды равна 1000. Исходная функция для триангуляции строится как $p(x, y, z) - c = 0$, где c — заданный порог плотности. Для эксперимента $c = 1150$. Снимки получены с области грудной полости, в этом случае видны только кости и отложения кальция (рис. 18).

Неявные функции задаются следующими формулами:

— сфера: $x^2 + y^2 + z^2 - 1 = 0$;

— однополосный гиперболоид: $x^2 + y^2 - z^2 - 0.25 = 0$;

— цилиндр: $x^2 + y^2 - 1 = 0$;

— поверхность третьего порядка: $x^3 + y^2 - 1 = 0$.

В представленной таблице результатов приняты следующие обозначения: MC — марширующие кубы, MPrisms — марширующие призмы, MP6 — марширующие пирамиды, MT6 — марширующие тетраэдры 6, MT5 — марширующие тетраэдры 5, Skala — алгоритм Скалы.

Параметр *aspect ratio*						
Поверхность	MC	MPisms	MP6	MT5	MT6	Skala
Сфера	9.06	10.58	13.81	10.07	9.04	5.47
Гиперболоид	8.65	11.38	15.63	10.12	10.36	7.44
Цилиндр	4.61	6.62	15.79	4.61	6.51	4.44
Пов. 3-го порядка	3.04	4.88	12.73	3.73	4.86	4.46
Снимки томографа	5.94	6.93	10.00	6.05	6.99	5.17
Параметр *deviation*						
Поверхность	MC	MPisms	MP6	MT5	MT6	Skala
Сфера	2.77×10^{-5}	2.61×10^{-5}	1.56×10^{-5}	2.36×10^{-5}	2.40×10^{-5}	1.21×10^{-5}
Гиперболоид	9.60×10^{-6}	9.81×10^{-6}	6.08×10^{-6}	8.90×10^{-6}	9.31×10^{-6}	4.90×10^{-6}
Цилиндр	1.75×10^{-5}	1.75×10^{-5}	9.91×10^{-6}	1.56×10^{-5}	1.58×10^{-5}	8.14×10^{-6}
Пов. 3-го порядка	1.78×10^{-5}	1.62×10^{-5}	9.93×10^{-6}	1.64×10^{-5}	1.64×10^{-5}	8.09×10^{-6}
Снимки томографа	6.41×10^{-3}	6.15×10^{-3}	4.73×10^{-3}	4.89×10^{-3}	4.67×10^{-3}	3.48×10^{-3}
Параметр *deviation norm*						
Поверхность	MC	MPisms	MP6	MT5	MT6	Skala
Сфера	3.06×10^{-3}	3.54×10^{-3}	4.10×10^{-3}	4.22×10^{-3}	4.17×10^{-3}	1.77×10^{-3}
Гиперболоид	4.58×10^{-3}	6.39×10^{-3}	5.38×10^{-3}	5.83×10^{-3}	6.28×10^{-3}	4.86×10^{-3}
Цилиндр	1.85×10^{-3}	1.84×10^{-3}	3.71×10^{-3}	2.84×10^{-3}	2.86×10^{-3}	1.86×10^{-3}
Пов. 3-го порядка	1.63×10^{-1}	1.56×10^{-1}	1.51×10^{-1}	1.58×10^{-1}	1.56×10^{-1}	1.53×10^{-1}
Снимки томографа	9.03×10^{-1}	9.03×10^{-1}	9.07×10^{-1}	9.04×10^{-1}	9.06×10^{-1}	9.06×10^{-1}

8. Выводы. В статье описаны алгоритмы триангуляции на основе алгоритмов "марширующие кубы" и "марширующие тетраэдры", развитие которых происходит независимо друг от друга. Предложены новые алгоритмы триангуляции, не относящиеся к данным классам и не уступающие по качеству получаемых треугольников. Показано, что построение триангуляции трехмерных объектов можно производить разбиением на произвольные фигуры, строя триангуляцию внутри каждой фигуры (в данной работе используются таблицы для определения треугольников внутри фигуры). Пространство можно разбивать на фигуры разных типов, например на кубы и пирамиды.

Для каждого из описанных методов подробно представлены особенности программной реализации.

Выполнен сравнительный анализ получаемых триангуляций для всех описанных методов. Наилучшим качеством треугольников по параметру *aspect ratio* обладают алгоритм "марширующие кубы" и алгоритм Скалы. Алгоритм Скалы генерирует в среднем в пять раз больше треугольников, чем алгоритм "марширующие кубы". Параметр *deviation* на три порядка хуже для томографических снимков в связи с тем, что аппроксимируемая поверхность не обладает гладкостью. Наилучшим по этому параметру обладает триангуляция, полученная алгоритмом Скалы. Параметр *deviation norm* ухудшается с увеличением порядка аппроксимируемой функции.

Развитием данной работы может быть разработка алгоритма триангуляции с использованием разбиения пространства на шестивершинники по сети ребер, построенной методом Скалы (в шаблоне будет три шестивершинника, а не 12 тетраэдров). Такая триангуляция будет содержать меньшее количество треугольников, но обладать такой же информативностью.

СПИСОК ЛИТЕРАТУРЫ

1. *Lorensen W.E., Cline H.* Marching cubes: a high resolution 3d surface construction algorithm // ACM SIGGRAPH Computer Graphics. 1987. **21**, N 4. 163–169.
2. *Gueziec A.* Exploiting triangulated surface extraction using tetrahedral decomposition // IEEE Transactions on Visualization and Computer Graphics. 1995. **1**, N 4. 328–342.
3. *Skala V.* Precision of iso-surface extraction from volume data and visualization // Electronic Proc. of the Conf. on Scientific Computing 2000. 368–378 (http://www.emis.de/journals/AMUC/_contributed/algo2000/skala.pdf).
4. *Carneiro B.P., Silva C.T., Kaufman A.E.* Tetra-Cubes: an algorithm to generate 3D isosurfaces based upon tetrahedra // Proc. of SIGGRAPH'96. Vol. 9. New Orleans, 1996. 205–210.
5. *Montani C., Scateni R., Scopigno R.* Discretized marching cubes // Proc. of the Conf. on Visualization'94. Washington, DC, 1994. 281–287.
6. *Bourke P.* Polygonising a scalar field. 1997 (http://astronomy.swin.edu.au/ pbourke/modelling/polygonise).
7. *Shephard M., Georges M.* Automatic three-dimensional mesh generation by the finite octree technique // Int. J. for Numerical Methods in Engineering. 1991. **32**. 709–749.
8. *Bloomenthal J.* An implicit surface polygonizer // Graphics Gems IV. Boston: Academic Press, 1994. 324–349.
9. *Natarajan B.K.* On generating topologically consistent isosurfaces from uniform samples // The Visual Computer: Int. J. of Computer Graphics. 1994. **11**, N 1. 52–62.
10. *Семенихин А., Игнатенко А.* Сравнительный анализ методов интерактивной триангуляции сеточных функций // Графика и мультимедиа. Вып. 6. 2004 (http://cgm.graphicon.ru/issue6/triangulation_comp/).
11. *Tavares G., Santos R., Lopes H., Lewiner T., Vieira A.W.* Topological reconstruction of oil reservoirs from seismic surfaces // Proc. of the Conf. of the Int. Association for Mathematical Geology. Vol. 1. Portsmouth, 2003. 27–34.
12. *Hoppe H.* Progressive meshes // Proc. of SIGGRAPH'96. New Orleans, 1996. 99–108.
13. *Evans F., Skiena S., Varshney A.* Efficiently generating triangle strips for fast rendering. Technical Report. Department of Computer Science. State Univ. of New York at Stony Brook, USA, 1997.
14. *Schroeder W.J., Zarge J., Lorensen W.E.* Decimation of triangle meshes // Computer Graphics. 1992. **26**, N 2. 65–70.

Поступила в редакцию
02.07.2007