УДК 681.3

БИНАРНЫЕ ОПЕРАЦИИ НАД ТРЕХМЕРНЫМИ ТЕЛАМИ

Н. А. Тюкачев

Воронежский государственный университет

Поступила в редакцию 01.03.2010 г.

Аннотация. Предлагается алгоритм определения принадлежности точки многоугольнику общего вида или многограннику с триангулированной поверхностью, основанный на локальном анализе семейств инцидентных вершин. Алгоритм, в отличие от ранее предлагаемых, не требует глобальной упорядоченности ребер и используется для оверлейных операций с трехмерными слоями в геоинформационных системах.
Ключевые слова: геометрические алгоритмы, алгоритмы на графах.

Abstract. the algorithm of definition of an accessory of a point Is offered to a polygon of a general view or a polyhedron with triangulation a surface, based on the local analysis of families of incidental tops. The algorithm, unlike earlier offered, does not demand global orderliness of edges and is used for overlay operations with three-dimensional layers in geoinformation systems.
Key words. geometrical algorithms, algorithms on graph.

В геоинформационных системах (ГИС) для описания слоев породы или рудного тела необходимо использовать трехмерные объекты. Эти объекты могут быть гомогенными, например, слой породы с одинаковой литологией или стратиграфией, или гетерогенными, например, рудное тело с плотностью руды, зависящей от координат.

Для описания гомогенных объектов можно использовать граничную модель, определяя границу как множество многоугольников. Частным случаем граничной модели является модель трехмерного тела, у которого граница представляет собой замкнутое множество нерегулярных треугольников. Замкнутую ориентированную поверхность P разобьем на треугольники со следующими условиями: а) каждая точка поверхности P принадлежит хотя бы одному треугольнику; б) два треугольника могут пересекаться только в одной вершине или по целому ребру. Такую модель будем называть *CTIN-представлением* или *CTIN-поверхностью* (CTIN — closed triangular irregular network). На рис. 1 изображены верхняя и боковая триангулированные поверхности слоя.

Первичные данные о таких поверхностях получают, как правило, бурением сотен скважин с взятием проб породы. За год компания АЛРОСА, например, бурит 2—3 сотни километров геологоразведочных скважин. Это порождает необходимость перерабатывать и изображать большой объем трехмерной информации.

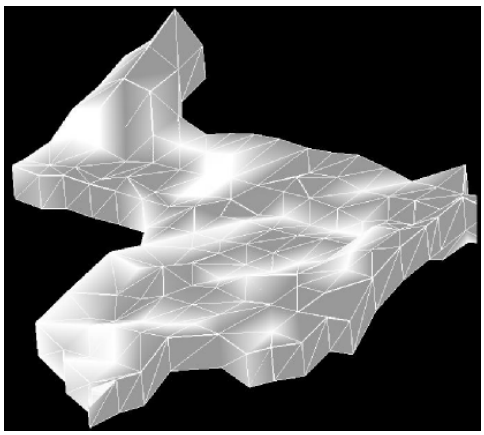

Рис. 1. Триангулированная поверхность слоя

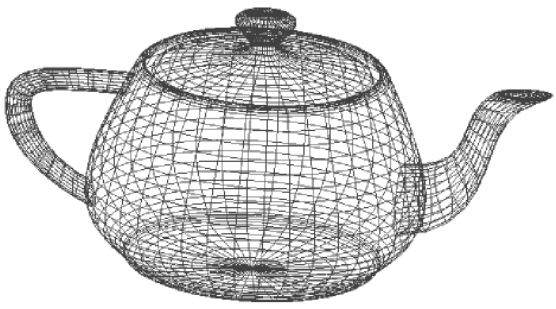

Рис. 2. Трехмерный объект, состоящий из четырех частей

Перечислим некоторые из задач, возникающих при работе с CTIN-поверхностями:

построение 2D-триангуляции верхней и нижней поверхности слоя;

построение 2D-триангуляции с ограничениями в виде произвольного многоугольника или ломаной линии, моделирующей разлом земной коры;

построение триангуляции всех слоев;

сглаживание триангуляции;

задача построения триангуляции боковой поверхности слоя, которая сводится к созданию упорядоченного по обходу массива номеров граничных точек;

построение триангуляции невыпуклого многоугольника вертикального разреза слоя;

построение изолиний;

булевы операции над 3D-телами, которые сводятся к задачам:

определение точек пересечения двух тел. Эта задача сводится к определению точек пересечения ребер одного тела с гранями второго тела;

разбиение ребер и треугольных граней на новые грани и ребра;

расстановка индексов для вершин, ребер и граней;

изображение тела.

В этой работе предлагается алгоритм для решения наиболее сложной восьмой задачи булевых операций с трехмерными телами. Необходимо отметить, что во многих графических библиотеках, например, OpenGL, DirectX, AutoCad Civil, эта задача не решается и обходится z-буферизацией. На рис. 2 приведено широко известное изображение чайника. Этот трехмерный объект состоит из четырех несвязанных между собой частей: сети четырехугольников, из которых составлена поверхность каждой части, не взаимодействуют между собой. Однако z-буферизация позволяет изображать эти части как единое целое.

Наша задача состоит в том, чтобы в результате любой булевой операции над двумя CTIN-поверхностями получить новую CTIN-поверхность. Рассмотрим в качестве примера два параллелепипеда A и B.

Поверхность первого параллелепипеда A разбита на 16 треугольных граней (рис. 4).

Поверхность второго параллелепипеда B также разбита на 16 треугольных граней (рис. 5). При пересечении этих двух тел добавляются новые вершины, ребра и грани (рис. 6).

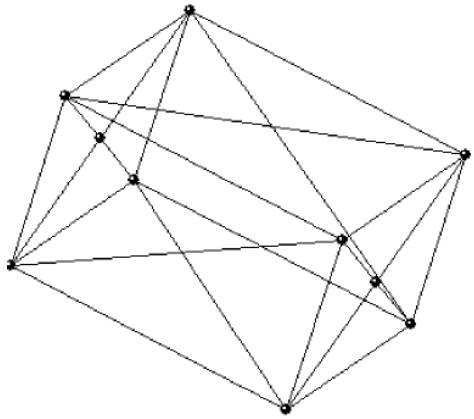

Рис. 4. Поверхность первого параллелепипеда

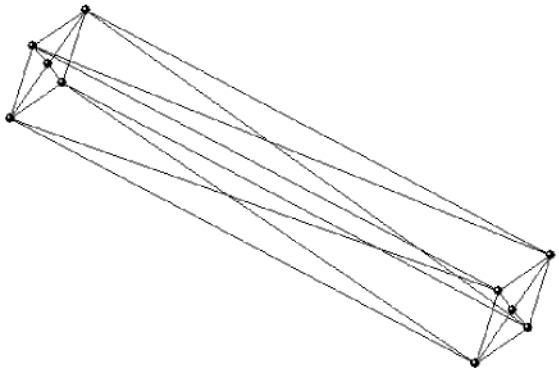

Рис. 5. Поверхность второго параллелепипеда

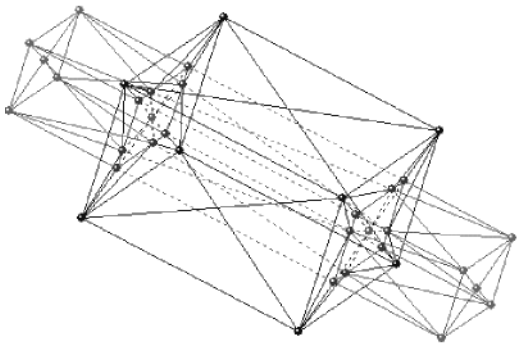

Рис. 6. Результат пересечения двух параллелепипедов

Новое тело C получается в результате одной из булевых операций, например $C = A \backslash B$ (рис. 9 и 10).

Обсудим более подробно понятие булевых операций. Из элементов двух множеств A и B пространства M, используя булевы операции объединения «$\cup$», пересечения «$\cap$» и разности «$\backslash$» и т.д., можно составить новое множество C (рис. 11).

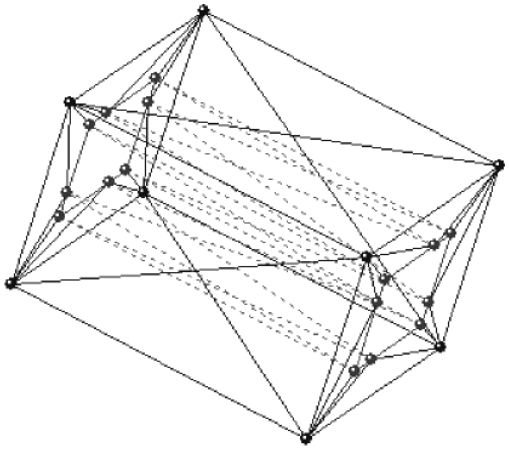

Рис. 7. Новое тело C=A\B (ребра)

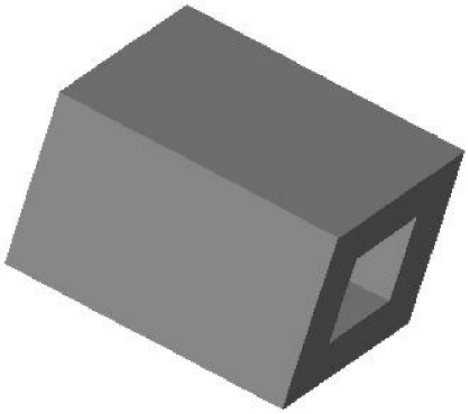

Рис. 8. Новое тело C=A\B (грани)

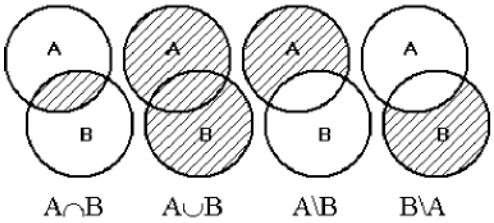

Рис. 9. Операции над множествами

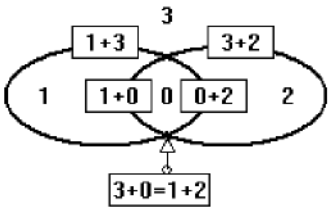

Рис. 10. Индексы подмножеств и границ при разбиении

Без ограничения общности можно считать, что $A \cap B \neq \varnothing$, то есть существуют x такие, что $(x \in A) \wedge (x \in B)$. Каждому элементу $x \in M$ поставим в соответствие число (рис. 12). Пронумеруем непересекающиеся части пространства M, в котором определены множества A и B: $A \cap B$ — 0; $A \setminus B$ — 1; $(A \cup B)$ — 2; $M \setminus (A \cup B)$ — 3.

Определение 1. Числа 0, 1, 2, 3 будем называть *индексами внутренних элементов множеств* $A \cap B$, $A \setminus B$, $B \setminus A$ и $M \setminus (A \cup B)$: Index = 0 : $A \cap B = \{x : x \in A$ и $x \in B\}$; Index = 1 : $A \setminus B = \{x : x \in A$ и $x \notin B\}$; Index = 2 : $B \setminus A = \{x : x \in B$ и $x \notin A\}$; Index = 3 : $A \cap B = \{x : x \notin A$ и $x \notin B\}$.

Далее вместо термина индекс внутреннего элемента множества будем употреблять термин индекс множества. Индексы множества A равны ненулевым битам в двоичном числе 0011_2, а индексы множества B равны ненулевым битам в числе 0101_2.

Определение 2. Каждому граничному элементу множеств A и B поставим в соответствие число, которое будем называть *индексом границы*. Индекс границы равен сумме индексов множеств, которые она разделяет (рис. 4): 1+0=1 — индекс граничной точки между множествами с индексами 0 и 1; 2+0=2 — индекс граничных точек между множествами с индексами 0 и 2; 3+0=1+2=3 — индекс узловых точек; 3+1=4 — индекс точки между множествами с индексами 1 и 3; 3+2=5 — индекс граничной точки между множествами с индексами 3 и 2.

В любой узловой точке — точке пересечения границ — сходятся две или четыре (например, для операции исключающее или с номером $6_{10}=0110_2$) границы нового множества C.

Операции объединения, вычитания, пересечения 2D и 3D тел являются достаточно сложными. Однако, бинарная нумерация операций над множествами позволяет сформулировать общий алгоритм сбора ребер и граней в результирующее тело, опирающийся на теорему об индексах [18, 19].

В общем случае над двумя множествами A и B можно рассматривать шестнадцать различных операций, каждая из которых имеет номер от 0 до 15. Для дальнейших рассуждений будем использовать двоичную нумерацию этих операций. Двоичный номер операции полностью определяет ее результат. Так, например, операция с номером $7_{10}=0111_2$ — это операция объединения «или».

Теорема об индексах [18, 19]. Между номерами операций и индексами граничных элементов результирующего тела существует связь: сумма номеров пар различных битов в двоичном представлении номера операции совпадает с индексами граничных элементов.

Доказательство сводится к проверке утверждения для всех 16-ти бинарных операций. Так, например, для операции $C = (A \setminus B) \cup (B \setminus A)$ с номером $6_{10}=0110_2$ граница состоит из элементов с номерами: 0+1, 2+3, 1+3, 2+3, 3+0=1+2. В двоичном числе 0110_2 различны следующие пары битов: (0,1), (0,2), (2,3), (1,3). Элементы пересечения границ (0+3=1+2) входят в границы любого нового множества. Для операции пересечения A∩B с номером $1_{10}=0001_2$ граница множества C состоит из элементов с индексами 0+1, 0+2, 0+3=1+2. Эти же пары различных битов есть в числе 0001_2: (0,1), (0,2), (0,3).

Замечание. Порядок нумерации множеств $A \cap B, A \setminus B, B \setminus A$ и $M \setminus (A \cup B)$ влияет на порядок нумерации булевых операций.

Таким образом, построение результирующего тела для любой булевой операции сводится к задаче определения принадлежности точки v телу.

Известно несколько алгоритмов определения принадлежности точки многоугольнику. Сложность проблемы заключается в том, что многоугольник может быть невыпуклым и содержать в себе «дырки». Как правило, алгоритмы требуют упорядоченности ребер по контуру, а эта задача решается нахождением эйлерова цикла на графе.

Перечислим некоторые известные алгоритмы определения принадлежности точки $Q(x,y)$ многоугольнику: алгоритм для выпуклых многоугольников [11], тригонометрический алгоритм, триангуляционный алгоритм [13], определение по площади для выпуклых многоугольников, лучевой алгоритм.

Основные проблемы при использовании лучевого алгоритма возникают при прохождении луча Q через вершины: для многоугольника общего вида (рис. 13) в вершине может сходиться несколько ребер и ребра могут совпадать с лучом. Алгоритм предлагает в этом случае засчитывать пересечение луча только с концом ориентированного ребра. Важным аспектом является глобальная по всему контуру упорядоченность ребер, обеспечивающая то, что в вершине не могут сходиться концами два последовательных ребра.

Как уже утверждалось ранее, для многогранников глобальная упорядоченность ребер и граней, то есть существование замкнутых циклов по графу ребер и граней, невозможна. Поэтому необходимо использовать иной алгоритм, предлагаемый автором [17], который будем называть *инцидентным*.

Инцидентный лучевой алгоритм для многоугольников. Для алгоритмов принадлежности точки особую сложность представляют вершины, степень которых больше 2. Для дальнейшего анализа введем новое понятие — семейство инцидентных вершин. В плоском случае через вершину v может проходить несколько фрагментов контура. Каждый фрагмент может содержать только два ребра E0 и E1 с нормалями N0 и N1, у которых внешние нормали не меняют ориентацию, то есть скалярное произведение векторных произведений должно быть больше нуля: $(E_0 \times N_0, E_1 \times N_1) > 0$. Для многогранников в трехмерном пространстве на вершину v может опираться любое количество граней, ребер и вершин. Анализ известных алгоритмов для многоугольников показал, что даже в двумерном случае эта задача далеко не тривиальна, если луч, выходящий из точки, проходит через вершину многоугольника.

Для описания триангулированной поверхности будем использовать модель «сущность-связь» (Entity-Relationship model или ER-модель). Рассмотрим многогранники общего вида, которые могут быть определены четырьмя сущностями: тела (Body), грани (Face), ребра (Edge) и вершины (Vertex). В общем случае эти сущности связаны между собой шестью отношениями «многие ко многим» *:* (тела-грани, тела-ребра, тела-вершины, грани-ребра, грани-вершины, ребра-вершины) и четырьмя отношениями «один ко многим» (тело — инцидентные тела, грань — инцидентные грани, ребро — инцидентные ребра, вершина — инцидентные вершины). Но для алгоритма достаточно оставить 11 связей «один ко многим» (рис. 17).

Обращаем внимание на то, что для реализации алгоритма для каждой вершины потребуется знать семейства ее соседей, то есть оставить связь «вершина — инцидентные вершины».

Структура сущности `Vertex` помимо координат `x, y, z`, массивов номеров граней `IdFace` и тел `IdBody`, содержит поле `Family` — двумер-

ный массив семейств номеров инцидентных вершин и поле `Visit` — признак посещения вершины, используемый при реализации алгоритма принадлежности. Этот признак запрещает обрабатывать вершину дважды.

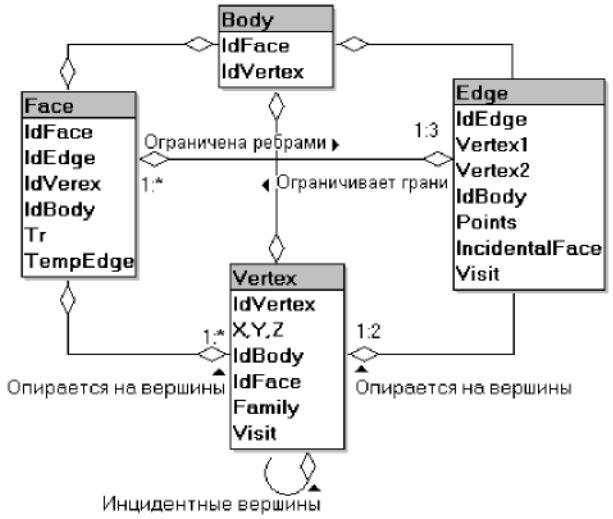

Рис. 15. ER-модель для многогранников

Структура ребер `Edge` содержит номера вершин `Vertex1` и `Vertex2`, на которые ребро опирается, массив `IdBody` номеров тел, которому ребро принадлежит, временный массив точек Points, которые добавляются при пересечении тел, и массив соседних граней `IncidentalFace`.

Структура треугольных граней `Face` содержит массивы `IdVertex` — номеров вершин, на который она опирается, массив `IdEdge` номеров ребер, составляющих контур грани и массив `IdBody` номеров тел, которому ребро принадлежит. Кроме этого у грани есть временный массив `TempEdge`, в который добавляются ребра пересечения с гранями другого тела, и временный массив `Tr` треугольников триангуляции.

Наконец, структура `Body` содержит массив номеров граней `IdFace` и вершин `IdVertex`.

Прежде, чем перейдем к краткому обзору алгоритмов бинарных операций над многоугольниками, сделаем одно замечание. Практически все алгоритмы вычисления пересечения, объединения или разности многоугольников, кроме алгоритмов Леонова, линейно-узлового и триангуляционного, реализуют набор операций, не являющийся замкнутым [20], так как в результате их выполнения могут получаться контуры с самокасаниями, что неприемлемо в качестве исходных данных для почти всех алгоритмов, и с отверстиями, которые, например, не допускаются алгоритмом Шутте [4]. Кроме этого, многие из широко известных алгоритмов не позволяют корректно обрабатывать вершины, в которых сходится более двух ребер многоугольников.

1. Алгоритм Сазерленда — Ходжмана [5] определяет только *пересечение* многоугольника *общего* вида с *выпуклым* многоугольником-отсекателем.

2. Алгоритм О'Рурка один из наиболее простых алгоритмов пересечения *выпуклых* многоугольников [3]. Предположим, что $(p_1,p_2,...,p_L)$ и $(q_1,q_2,...,q_M)$ — это списки вершин P и Q, упорядоченные при обходе их против часовой стрелки двух выпуклых многоугольников. Для каждого многоугольника объявляются текущие вершины p_i и q_i, кроме этого, текущие ребра $E(p_{i-1}p_i)$ и $E(q_{i-1}q_j)$. Алгоритм заключается в том, чтобы не двигаться по той границе P или Q, текущее ребро которой еще может содержать необнаруженную точку пересечения. Анализируется четыре возможных варианта взаимного расположения вершин p_i и q_j и ребер $E(p_{i-1}p_i)$ и $E(q_{j-1}q_j)$.

3. Алгоритм Вейлера-Азертона [6, 7, 12] является гораздо более мощным алгоритмом отсечения многоугольников по сравнению с ранее перечисленными. Применим для простых областей с «дырами» без самопересечений и для любых регулярных отсекателей. Алгоритм описывает оба многоугольника как циклические списки вершин. Внешние границы многоугольников представляются упорядоченными по часовой стрелке, а внутренние границы и «дыры» упорядоченными против часовой стрелки. При перемещении по списку вершин, внутренняя область многоугольника должна всегда находиться справа.

4. Алгоритм Леонова [10]. В качестве исходных данных могут быть произвольные полигональные многосвязные области с отверстиями и неограниченной кратностью вершин. Алгоритм состоит из четырех шагов.

На *первом* шаге находятся все пары несмежных пересекающихся ребер регионов A и B. Если точки пересечения не совпадают с концевыми точками ребер, в исходные регионы добавляются новые вершины с координатами соответствующих точек.

На *втором* шаге рассматривается некоторый ограничивающий контур C одного из регионов

A и B. За M обозначается регион, к которому не принадлежит C. На предыдущем шаге все точки пересечения были добавлены в исходные регионы. Пометка контура C или ребра, принадлежащего контуру C, делается в зависимости от геометрического положения относительно M.

На *третьем* шаге последовательно рассматривается каждый контур из регионов A и B. В зависимости от метки, контур включается в результат с нужным направлением или совершается обход всех ребер C, чтобы найти ребра, с которых начнется сборка результирующих контуров.

На *четвертом* шаге происходит формирование результирующего контура R. Для каждого внешнего контура создается отдельный полигон, внешним контуром которого он и будет являться. Внутренние контуры на предыдущем этапе сохраняются во временном списке, а затем помещаются в результирующие полигоны. Регион R будет состоять из множества полученных таким образом полигонов.

5. Алгоритм Шутте [4] основан на алгоритме Вейлера-Азертона [6], однако имеет несколько ограничений: 1) вершины многоугольников должны быть упорядочены по часовой стрелке; 2) многоугольники не должны иметь «дыр»; 3) многоугольники не должны быть самопересекающимися.

Алгоритм состоит из следующих шагов:

Шаг 1. Пересечение двух многоугольников. Результатом являются те же самые многоугольники, с одним отличием, что точки пересечения добавляются как вершины многоугольников.

Шаг 2. Все ребра обоих многоугольников помечаются как «внутренние» (ребро находится внутри другого многоугольника), «разделенные» (ребро принадлежит обоим многоугольникам) и «наружные» (ребро находится снаружи другого многоугольника).

Шаг 3. Находятся минимальные многоугольники.

Шаг 4. Все минимальные многоугольники классифицируются по трем выходным наборам $A \cap B, , B/A, A/B$. *Конец алгоритма.*

Процесс классификации минимальных многоугольников происходит в два этапа. На первом этапе проверяется «отцовство» каждого минимального многоугольника, после этого решается, к какому классу относится минимальный многоугольник. Ответ на вопрос, является ли минимальный многоугольник потомком исходного многоугольника, производится при помощи маркировки ребер. Для каждого ребра минимального многоугольника выясняется родительский исходный многоугольник.

6. Алгоритм Холверда. Входными данными *скан-линейного* алгоритма Холверда [1] являются два набора многоугольников. Идея алгоритма состоит в том, чтобы разбить плоскость на секции, называемые *закрытыми областями*, описываемыми простыми многоугольниками. Результат булевой операции может быть получен, если определена принадлежность каждой из закрытых областей одному или обоим наборам многоугольников, для этого необходимо вычислить пересечения всех сегментов многоугольников.

7. Алгоритм Маргалита-Кнотта [2] делится на две основные стадии. Первая стадия — это классификация линейных сегментов входных многоугольников, а вторая — конструирование результирующих многоугольников.

Основные этапы алгоритма:

Шаг 1. Классифицирует вершины каждого многоугольника на нахождение внутри, снаружи или на границе другого многоугольника.

Шаг 2. Ищутся все точки пересечения между многоугольниками. Для каждого многоугольника вершины вместе с точками пересечения помещаются в такую структуру данных, что две соседние точки определяют оригинальное ребро или часть оригинального ребра результирующего многоугольника.

Шаг 3. Классифицирует реберные фрагменты одного многоугольника на нахождение внутри, снаружи или на границе другого многоугольника. Множество реберных фрагментов многоугольника Q делится на три подмножества, в зависимости от положения относительно многоугольника P.

Шаг 4. Классифицированные реберные фрагменты сохраняются в структуре данных, позволяющей быстро выполнять операции поиска и удаления. После создания каждого результирующего многоугольника его вершины помещаются в выходной массив, а его реберные фрагменты удаляются из структуры данных для того, чтобы приготовиться для конструирования следующего результирующего многоугольника.

Каждый результирующий многоугольник создается при помощи последовательного поиска следующего реберного фрагмента, пока

не будет найден фрагмент, вторая конечная точка которого посещается во второй раз. На этой точке результирующий многоугольник найден.

8. Триангуляционный алгоритм Скворцова. Идея алгоритма построения оверлеев многоугольников с помощью триангуляции [16] заключается в построении триангуляции с ограничениями ребер исходных многоугольников, а затем объединения некоторых треугольников в результирующий многоугольник.

9. Линейно-узловой алгоритм Скворцова [15]. Алгоритм состоит трех шагов:

Шаг 1. Построение планарного графа, ребра которого должны соответствовать ребрам границ многоугольников.

Шаг 2. Классификация ребер графа по признаку вхождения в результирующий многоугольник. Для классификации ребер требуется выполнить две операции:

1) определить расположение многоугольника относительно ребра;

2) определять, попадает ли некоторое ребро на границу, внутрь или вне многоугольника.

Шаг 3. Выполняется сборка ребер графа в последовательность ребер границы результирующего многоугольника. Для этого необходимо пометить входящие в результат ребра как непройденные, а остальные — как пройденные и запустить алгоритм выделения контуров.

В результате работы предложенного алгоритма будет построен многоугольник, в котором контуры не будут самопересекаться, и каждый контур будет задан в таком порядке обхода точек, что многоугольник всегда находится справа по ходу обхода.

10. Индексный алгоритм. В работах автора [18, 19] предложен алгоритм, основанный на понятии индекса вершин и границ. Алгоритм применим для многоугольников и многогранников общего вида:

Шаг 1. Для всех вершин многоугольников A и B определить соседние вершины и ребра;

Шаг 2. Решая задачу принадлежности вершин одного тела другому, расставить индексы вершин (1 5);

Шаг 3. Упорядоченно добавить точки пересечения ребер многоугольников A и B;

Шаг 4. Разбить ребра добавленными точками.

После выполнения трех шагов алгоритма выполняются следующие условия:

1. Ребра многоугольников A и B не имеют никаких общих точек друг с другом, кроме концевых.

2. Ребра многоугольников A и B могут находиться целиком (за исключением, может быть, концевых точек) внутри или вне области, описываемых A и B.

Шаг 5. Расставить индексы ребер и по номеру булевой операции на основе теоремы об индексах собрать в новый многоугольник C ребра с подходящими индексами.

Достоинства: алгоритм замкнут; применим для многоугольников и многогранников общего вида; прост на заключительной стадии сбора результирующего мгогоугольника.

На рис. 18 приведен результат выполнения операции «и».

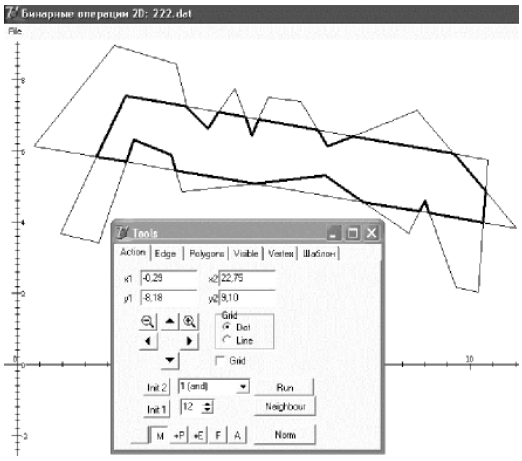

Рис. 16. Операция «и» для двух полигонов

Алгоритм построения результирующего **трехмерного тела** для булевой операции состоит из следующих семи шагов:

Определить индексы вершин 1-го и 2-го многогранников;

Найти пересечение всех граней 1-го многогранника со всеми гранями 2-го многогранника;

Найти пересечение грани 1-го многогранника с ребрами грани 2-го многогранника;

Если точка пересечения ребра принадлежит грани, то ищем ее в массиве вершин Vertex, если не найдена, то добавляем ее с номером n и добавляем в массив t[];

Добавляем номер точки n в массив Points ребра;

Добавляем номер точки n в массив Face1. IdVertex;

Добавляем номер точки n в массив Face2. IdVertex;

Найти пересечение грани 2-го многогранника с ребрами грани 1-го многогранника;

Если точка пересечения ребра принадлежит грани, то ищем ее в массиве вершин Vertex, если не найдена, то добавляем ее с номером n и добавляем в массив t[];

Добавляем номер точки n в массив Points ребра;

Добавляем номер точки n в массив Face1. IdVertex;

Добавляем номер точки n в массив Face2. IdVertex;

Если пересечение грани с гранью добавляет 2 точки (длина массива t[] равна 2), то в массив грани TempEdge добавляем номера этих точек;

Триангуляция всех граней 1-го многогранника по добавленным вершинам с ограничением не пересекать ребра TempEdge. Для каждой грани создается массив треугольников Tr;

Триангуляция всех граней 2-го многогранника по добавленным вершинам с ограничением не пересекать ребра TempEdge. Для каждой грани создается массив треугольников Tr;

Добавление ребер;

Создать копию массива ребер Ed;

Для всех ребер, если на них есть новые точки в массиве Points, то отсортировать их;

Разбить ребро точками Points и добавить их в массив Ed;

Добавить ребра из треугольников триангуляции Tr;

Временный массив Ed переместить в массив ребер;

Добавление граней;

Создать копию массива граней Fa;

Добавить в массив Fa треугольники триангуляции Tr;

Временный массив Fa переместить в массив граней;

Добавление нового многогранника;

Если вершина подходит по индексу, то добавить его в новый многогранник;

Расставить индексы ребер;

Если ребро подходит по индексу, то добавить его в новый многогранник;

Расставить индекса граней;

Если грань подходит по индексу, то добавить ее в новый многогранник.

Задачу определение индекса точки, то есть определение принадлежности точки телу необходимо выполнять на первом шаге и на шагах 7.3 и 7.5.

Опишем более подробно второй шаг: пересечение грани с гранью. На шаге 2.1 определяется точка пересечения ребра и грани и добавляется в массив Points ребра. На шаге 5.2 и 5.3 эти точки будут использованы для разбиения ребра на новые ребра. Если в результате пересечения грани с гранью добавилось две таких точки (а их всегда 0 или 2), то в массиве TempEdge граней необходимо зафиксировать ребро пересечения граней. Это ребра не имеет права пересекать триангуляция граней, выполняемая на шагах 3 и 4. Номера точек пересечения добавляются также в массивы граней Face1. IdVertex и Face2.IdVertex. Эти точки используются при построении триангуляции граней.

На 3 и 4 шагах выполняется триангуляция граней с ограничениями. На рис. 19 показана пирамида с вершинами 0—1—2—3, ребра которой пересеклись с гранями другой пирамиды по точкам 8, 9 и 10. В результате некоторые боковые грани пришлось разбить на три новых грани.

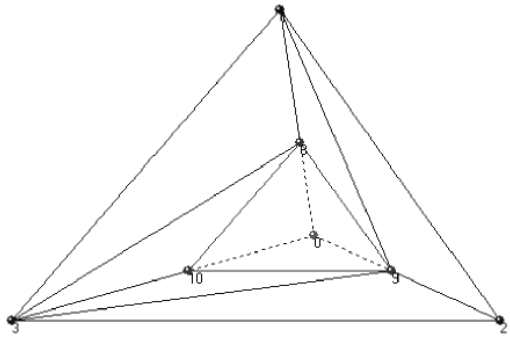

Рис. 17. Пирамида с добавленными точками и ребрами

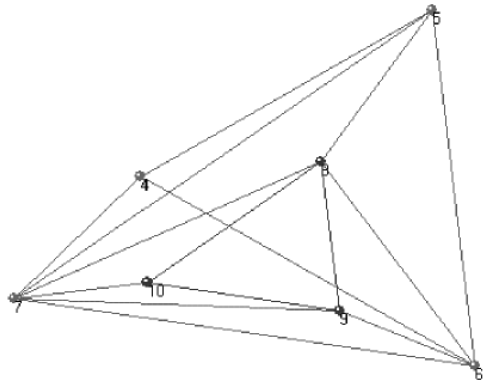

Рис. 18. Один из вариантов разбиения грани 5—6—7

Задача разбиения на треугольники и добавления новых ребер в общем случае не тривиальна. На рис. 20 показан случай, когда в грань 5—6—7 пирамиды 4—5—6—7 по треугольнику 8—9—10 «врезалось» второе тело. Грань 5—6—7 приходится разбивать на семь треугольников.

Эта задача сводится к построению триангуляции по заданным точкам, но при этом недопустима «правильная» в смысле Делоне триангуляция, если хотя бы одна из сторон треугольника пересекает полигон 3—4—5 пересечения грани со вторым телом, как это показано на рис. 21.

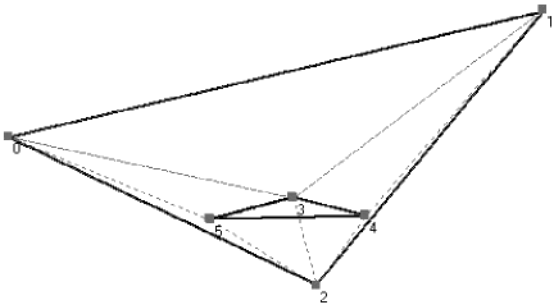

Рис. 19. Недопустимая триангуляция грани

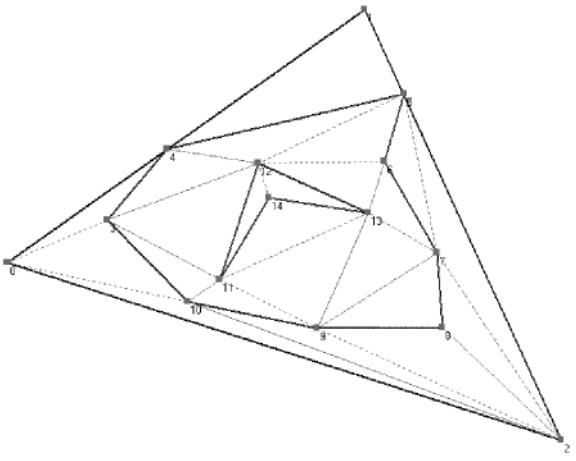

Рис. 20. Невыпуклая неодносвязная область пересечения грани с телом

В общем случае область песечения грани с телом может быть и невыпуклая неодносвязная (рис. 22). Более подробное обсуждение этой проблемы можно найти в [14].

Выполнение 5 и 6 шагов особых проблем не вызывает. Обсудим более подробно выполнение последнего, седьмого, шага алгоритма: добавление нового многогранника.

СПИСОК ЛИТЕРАТУРЫ

1. *Holwerda K.* Complete Boolean Description (http:// www.xs4all.nl/~kholwerd/bool.html).

2. *Margalit A., Knott G. D.* An algorithm for computing the union, intersection or difference of two polygons // Computers & Graphics. 1989. V. 13. No. 2. P. 167—183.

3. *O'Rourke J., Chien C. B., Olson T., Naddor D.* A new linear algorithm for intersecting convex polygons // Computer Graphics and Image Processing. 1982. V. 19. P. 384—391.

4. *Schutte K.* An edge labeling approach to concave polygon clipping // ACM Transactions on Graphics. 1995. P. 1—10.

5. *Sutherland I. E., Hodgman G. W.* Reentrant polygon clipping // CACM. 1983. V. 26. P. 868—877.

6. *Weiler K.* Polygon comparison using graph representation // Computer Graphics. 1980. V. 14. P. 10—18.

7. *Weiler K., Atherton P.* Hidden surface removing using polygon area sorting // Computer Graphics. 1977. V. 11. P. 214—222.

8. *Конноли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. / Томас Конноли, Карелии Бегг. — М.: Издательский дом "Вильямс", 2003. — 1440 с.

9. *Ласло М.* Вычислительная геометрия и компьютерная графика на С++. / М. Ласло — М.: БИНОМ, 1997.

10. *Леонов М. В., Никитин А. Г.* Эффективный алгоритм, реализующий замкнутый набор булевых операций над множествами многоугольников на плоскости / Препринт Института систем информатики СО РАН № 46. 1997. — 20 с.

11. *Рвачев В. Л.* Методы алгебры логики в математической физике. / В. Л. Рвачев. — Киев: Наукова думка, 1974. — 259 с.

12. *Роджерс Д.* Алгоритмические основы машинной графики / Пер. с англ. М.: Мир, 1989. — 512 с.

13. *Скворцов А. В.* Алгоритмы построения и анализа триангуляции. / А. В. Скворцов, Н. С. Мирза. — Томск: Изд-во Том. ун-та, 2006. — 167 с.

14. *Скворцов А. В.* Алгоритмы построения триангуляции с ограничениями // Вычислительные методы и программирование. 2002. № 3. С. 82—92 (http://num-meth.srcc.msu.su).

15. *Скворцов А. В.* Линейно-узловой алгоритм построения оверлеев двух полигонов // Вестник ТГУ. 2002. № 275. С. 99—103.

16. *Скворцов А. В.* Построение объединения, пересечения и разности произвольных многоугольников в среднем за линейное время с помощью триангуляции // Вычислительные методы и программирование. 2002. Т. 3. С. 116—123.

17. *Тюкачев Н. А.* Определение принадлежности точки многоугольнику общего вида методом трассировки луча // — Воронеж. Вестник ВГТУ. 2009. № 5. С. 338—345.

18. *Тюкачев Н. А., Илларионов И. В., Хлебостроев В. Г.* Программирование графики в Delphi. СПб., БХВ-Петербург. 2008. — 766 с.

19. *Тюкачев Н. А., Свиридов Ю. Т.* Создание мультимедийных приложений в Delphi. СПб., Питер. 2001. — 400 с.

20. *Ченцов О. В., Скворцов А. В.* Обзор алгоритмов построения оверлеев многоугольников // — Томск. Вестник ТГУ, 2003, № 280. — С. 338—345.

Тюкачев Николай Аркадьевич — к. ф.-м. н., доц. , зав. каф. Программирования и информационных технологий, Воронежский государственный университет.Тел. (4732) 208-470. E-mail: nik@cs.vsu.ru

Tjukachev N.A. — Candidat of Physics-math. Sciences, Associate Professor, the dept. of the Programming and Information Technologies, Voronezh State University. Tel. (4732) 208-470. E-mail: nik@cs.vsu.ru