УДК 519.688

ПОСТРОЕНИЕ ОБЪЕДИНЕНИЯ, ПЕРЕСЕЧЕНИЯ И РАЗНОСТИ ПРОИЗВОЛЬНЫХ МНОГОУГОЛЬНИКОВ В СРЕДНЕМ ЗА ЛИНЕЙНОЕ ВРЕМЯ С ПОМОЩЬЮ ТРИАНГУЛЯЦИИ

А. В. Скворцов[1]

Рассматривается применение триангуляции с ограничениями для построения оверлеев произвольных многоугольников. Приводится сравнение с другими алгоритмами.

Введение. Задача построения оверлеев (объединения, пересечения и разности) произвольных многоугольников (включая многоконтурные) часто возникает в системах автоматизированного проектирования (САПР), геоинформационных системах (ГИС), в других графических системах. На основе алгоритма построения оверлеев решается целый ряд других смежных задач (например, построение буферных зон и построение зон близости) [1, 4].

В настоящее время для решения данной задачи предложено большое количество алгоритмов. Тем не менее, на практике почти для всех из них можно найти примеры исходных данных, для которых алгоритмы не работают или работают некорректно. Это связано с двумя основными причинами. Во-первых, многие алгоритмы имеют ограничения на тип исходных данных (например, обрабатывают только выпуклые или одноконтурные многоугольники). Во-вторых, здесь очень остро стоит проблема вычислительной точности. Многие алгоритмы, для которых теоретически строго доказана правильность работы, в действительности могут оказаться бесполезными в силу ограниченной точности представления вещественных чисел в памяти компьютеров и потери точности в промежуточных вычислениях [1].

В настоящей работе развивается алгоритм построения оверлеев с помощью триангуляции, разработанный автором ранее [4]. Предлагается новый алгоритм классификации треугольников с меньшей трудоемкостью. Это позволяет получить практичный алгоритм, позволяющий строить оверлеи произвольных многоконтурных многоугольников за время, в среднем линейное относительно количества точек в исходных многоугольниках и количества точек пересечения многоугольников.

1. Определения. Вначале дадим определения возможных типов многоугольников, которые могут подаваться на вход алгоритмов построения оверлеев. Будем считать, что все многоугольники расположены на евклидовой плоскости.

Определение 1. *Границей многоугольника* G будем называть множество контуров $\{K_1, \ldots, K_m\}$, $m \geqslant 1$, каждый из которых является замкнутой ломаной линией, соединяющей набор точек на плоскости (вершин многоугольников): $K_i = \left((x_1^i, y_1^i), (x_2^i, y_2^i), \ldots, (x_{n_i}^i, y_{n_i}^i), (x_1^i, y_1^i)\right)$, $n_i \geqslant 3$.

Заметим, что в соответствии с этим определением допускаются взаимные пересечения и самопересечения контуров.

Определение 2. Пусть даны некоторая граница G и некоторая точка (x, y), не лежащая на границе G. Проведем через точку (x, y) любой луч, не проходящий через узлы границы G. Будем считать, что точка (x, y) находится *внутри* границы G, если число пересечений луча с отрезками ломаных границы нечетно.

Данное определение является конструктивным; его внутренняя непротиворечивость показывается на основе теоремы Жордана о замкнутой кривой, например, в [1].

Определение 3. *Многоугольником* P будем называть геометрическое место точек, лежащих на заданной границе G либо находящихся внутри нее (рис. 1). Отрезки ломаных границы G будем называть *сторонами* многоугольника.

В литературе так определяют общие многоугольники, чтобы отличить их от более простых частных случаев [1, 5].

Определение 4. Многоугольник называется *выпуклым*, если определяемое им множество точек является выпуклым (рис. 2 (а)).

Определение 5. *Простым* называется многоугольник, в границе которого отсутствуют взаимные пересечения и самопересечения контуров (рис. 2 (б)).

[1] Томский государственный университет, факультет информатики, пр. Ленина, д. 36, 634050, г. Томск; e-mail: skv@csd.tsu.ru

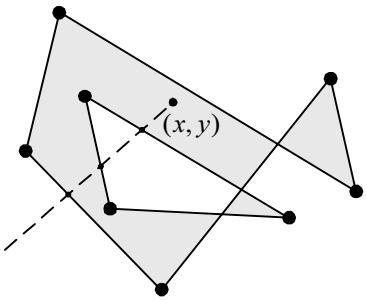

Рис. 1. Определение многоугольника

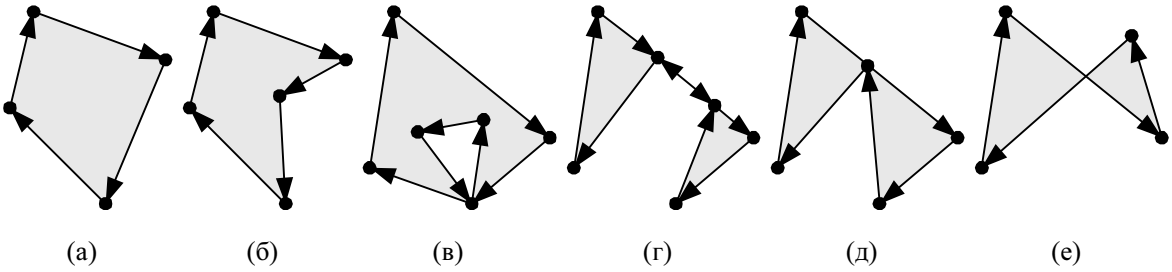

Рис. 2. Виды многоугольников: (а) — простой выпуклый, (б) — простой, (в) — регулярный, (г) — вершинно-полный, (д) — ориентируемый, (е) — общий

Определение 6. *Регулярным* называется многоугольник, в границе которого любая пара отрезков ломаных, определяющих границу многоугольника, либо не пересекается, либо пересекается только своими концами (рис. 2 (в)).

Определение 7. *Вершинно-полным* называется многоугольник, в границе которого любая пара отрезков ломаных, определяющих границу многоугольника, либо не пересекается, либо пересекается только своими концами, либо эти отрезки совпадают как множества точек (рис. 2 (г)).

Определение 8. *Ориентируемым* называется многоугольник, в границе которого любая пара отрезков ломаных, определяющих границу многоугольника, либо не пересекается, либо пересекается, при этом во множество точек пересечения входит хотя бы один конец этих отрезков (рис. 2 (д)).

Определение 9. *Вырожденным* называется многоугольник, в котором имеются стороны нулевой длины или имеется пересечение некоторых двух сторон, содержащее более одной точки плоскости (например, таковым является многоугольник на рис. 2 (г), т.к. в нем имеются две совпадающие стороны).

Теперь сформулируем задачу построения оверлеев. Пусть заданы два многоугольника P_1 и P_2 в виде границ G_1 и G_2. Требуется найти границы многоугольников I, U и D, определенных как соответствующие булевы операции над множеством всех точек исходных многоугольников:

$$I = \overline{\breve{P}_1 \cap \breve{P}_2}, \quad U = \overline{\breve{P}_1 \cup \breve{P}_2}, \quad D = \overline{\breve{P}_1 - \breve{P}_2},$$

где знаком "$\breve{}$" обозначена операция размыкания (вычитание из множества его границы), а знаком "$\overline{\ }$" — замыкание множества. При этом в качестве решения обычно требуется построить регулярный многоугольник.

Использование операций замыкания и размыкания в постановке задачи позволяет избежать некоторых вырожденных ситуаций при построении оверлеев. Иначе, например в случае нахождения пересечения двух касающихся многоугольников, может образоваться множество в виде линии или даже одной точки, которое нельзя представить в форме невырожденного многоугольника.

В дальнейшем при анализе трудоемкости алгоритмов построения оверлеев мы будем использовать следующие параметры: n_1 — количество вершин многоугольника P_1; n_2 — количество вершин многоугольника P_2; n_0 — количество точек пересечения границ G_1 и G_2; $n = n_1 + n_2 + n_0$.

2. Обзор известных алгоритмов. Задача построения оверлеев впервые возникла в машинной

графике как задача отсечения невидимого изображения, т.е. как задача вычисления пересечения двух многоугольников. В связи с тем, что в машинной графике область отсечения обычно является выпуклой областью, в настоящее время имеется достаточное количество алгоритмов построения оверлеев для случая, когда один из многоугольников является выпуклым.

2.1. Алгоритм О'Рурка. Одним из первых эффективных алгоритмов построения пересечения двух выпуклых многоугольников был предложен О'Рурком в [6]. Несмотря на то, что алгоритм изначально был предназначен для построения пересечения, его легко можно изменить для построения объединения и разности многоугольников.

В алгоритме О'Рурка предполагается, что вершины в обоих многоугольниках упорядочены в порядке обхода многоугольников против часовой стрелки. В каждом многоугольнике берется по одной вершине, которые становятся текущими p_i и q_j в выполняемом далее обходе (рис. 3 (а)). Затем выполняется цикл в $2(n_1 + n_2)$ шагов. На каждом шаге анализируется взаимное расположение текущих вершин p_i и q_j, а также сторон $\overline{p_i p_{i-1}}$ и $\overline{q_j q_{j-1}}$. На рис. 3 (б) – (д) приведены четыре из восьми возможных вариантов взаимного расположения (четыре других варианта соответствуют случаям, зеркальным по отношению к приведенным с заменой p_i на q_j и наоборот); штриховкой обозначена область, внутри которой целиком находится искомый многоугольник.

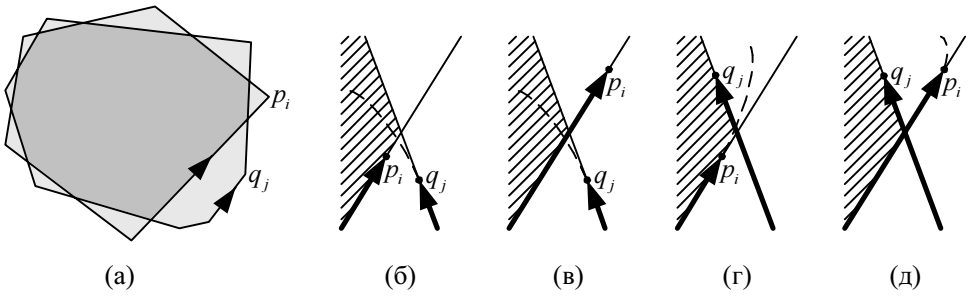

Рис. 3. Алгоритм О'Рурка построения оверлеев выпуклых многоугольников

На каждом шаге алгоритма необходимо определить, по какому из двух полигонов необходимо продвинуться вперед на одну вершину. Для первых двух случаев (как на рис. 3 (б),(в)) необходимо продвинуться на один шаг по второму многоугольнику, а для других случаев (как на рис. 3 (г),(д)) нужно двигаться по первому многоугольнику. Кроме того, для случая, показанного на рис. 3 (д), необходимо вычислить точку пересечения $\overline{p_i p_{i-1}}$ и $\overline{q_j q_{j-1}}$.

В результате выполнения обходов многоугольников находятся точки пересечения многоугольников, а также выявляются ломаные между найденными точками пересечения, которые должны попасть в результат. Для нахождения пересечения многоугольников нужно брать ломаные, лежащие левее в порядке обхода. Для нахождения объединения — ломаные, лежащие правее. Для нахождения разности нужно брать по две ломаные между точками пересечения, из которых правая ломаная принадлежит первому многоугольнику.

2.2. Алгоритм Сазерленда–Ходжмана. Алгоритм Сазерленда–Ходжмана позволяет вычислять только пересечение многоугольников, при этом один — *отсекающий* — многоугольник должен быть выпуклым, а другой — *отсекаемый* — может быть общим многоугольником [7].

Суть алгоритма заключается в последовательном отсечении отсекаемого многоугольника полуплоскостями, образованными сторонами отсекающего многоугольника. На каждом шаге отсечения полуплоскостью выполняется полный обход вдоль границы отсекаемого многоугольника, при этом находятся точки пересечения с прямой, определяющей полуплоскость, точки сортируются вдоль этой прямой и формируется граница нового многоугольника (рис. (4)).

Главными недостатками этого алгоритма являются невозможность его обобщения для построения объединения и разности многоугольников, а также относительно высокая трудоемкость, составляющая $O(n_1 n_2 \log n_2)$.

2.3. Алгоритм Вейлера–Азертона. Изначально алгоритм Вейлера–Азертона был предназначен для построения пересечения многоугольников [8], однако он может быть легко обобщен для построения объединения и разности.

Исходными данными в алгоритме Вейлера–Азертона могут быть только регулярные многоуголь-

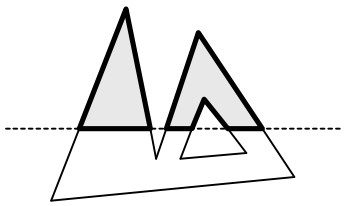

Рис. 4. Операция отсечения полуплоскостью в алгоритме Сазерленда–Ходжмана

ники, имеющие произвольное количество контуров. При этом порядок задания вершин в контурах должен быть таким, что при движении вдоль контуров внутренняя область многоугольника должна всегда находиться справа. Другими словами, для внешних контуров вершины должны задаваться по часовой стрелке, а для внутренних (дырок) — против (рис. 5 (а)).

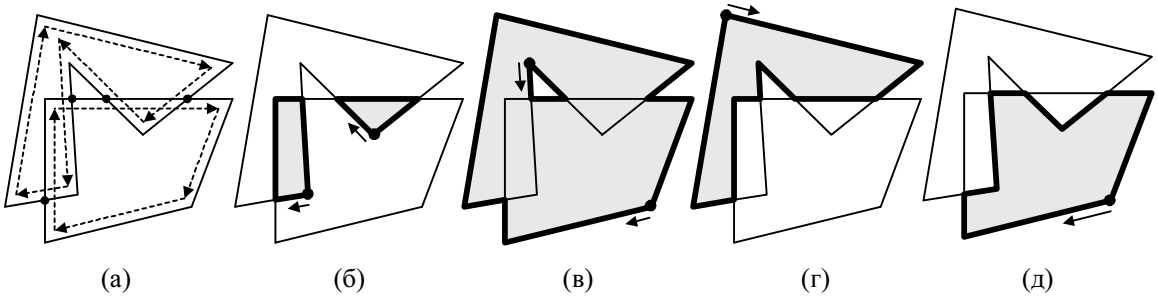

Рис. 5. Схема работы алгоритма Вейлера–Азертона: (а) — исходные многоугольники и точки их взаимного пересечения, (б) — поиск пересечения, (в) — объединения, (г) и (д) — разности многоугольников

Вначале в алгоритме определяются точки взаимного пересечения сторон многоугольника, а затем выполняется обход вдоль границ многоугольников для вычисления искомого многоугольника.

В задаче поиска пересечения обход начинается с любой вершины первого многоугольника, лежащей внутри второго. Обход идет вдоль границы первого многоугольника в порядке исходного задания вершин до точки пересечения со вторым. Здесь необходимо перейти на границу второго многоугольника и продолжить по ней движение в направлении, обратном порядку исходного задания вершин второго многоугольника. Обход выполняется до тех пор, пока путь не замкнется. Если есть еще не пройденные вершины, обходы нужно продолжить (рис. 5 (б)).

В задаче поиска объединения обход делается аналогично, но начинается с любой вершины первого многоугольника, лежащей вне второго. При этом движение по контурам выполняется в направлениях, соответствующих порядку исходного задания вершин многоугольников (рис. 5 (в)).

В задаче поиска разности обход делается аналогично объединению — с вершины первого многоугольника, лежащей вне второго. При этом движение по контурам выполняется в направлениях, как при поиске пересечений (рис. 5 (г),(д)).

Главными недостатками этого алгоритма являются необходимость учета большого количества особых случаев, возникающих при реализации и не оговоренных в оригинальной работе [8], и, как следствие, низкая вычислительная устойчивость. Кроме того, недостатком является ограничение на вид исходных данных.

2.4. Алгоритм Маргалита–Кнотта. Алгоритм Маргалита–Кнотта является развитием предыдущего алгоритма. Он позволяет обрабатывать вершинно-полные многоугольники и гораздо более устойчив в работе (см. [5]).

Алгоритм состоит из нескольких шагов. Вначале выполняется нормализация порядка задания в контурах многоугольников. Для вычисления объединения и пересечения порядок задания вершин делается таким, чтобы при обходе внутренности многоугольников находились всегда справа. Для вычисления разности порядок задания делается таким, чтобы при обходе внутренность первого многоугольника находилась всегда справа, а при обходе второго — слева.

Далее выполняется классификация всех вершин многоугольников относительно другого многоугольника. При этом возможны три варианта: вне, внутри или на границе. Затем находятся все точки взаимного пересечения, которые затем вносятся в многоугольники. После этого выполняется классификация сторон многоугольников относительно другого многоугольника.

В заключение выполняется обход вдоль классифицированных сторон многоугольников и собирается результирующий многоугольник по правилам, аналогичным применяемым в алгоритме Вейлера–Азертона.

Недостатком работы алгоритма Маргалита–Кнотта является достаточно большая трудоемкость, а также ограничение на тип исходных данных. Тем не менее, данный алгоритм достаточно широко используется на практике.

3. Алгоритм на основе триангуляции. Основная идея алгоритма построения оверлеев многоугольников с помощью триангуляции заключается в построении триангуляции с ограничениями, где в качестве структурных линий выступают стороны исходных многоугольников, а затем объединения некоторых треугольников в искомый многоугольник.

Отметим, что при построении триангуляции автоматически решается задача поиска точек пересечения и разбиения сторон многоугольников, возникающая во всех других алгоритмах построения оверлеев.

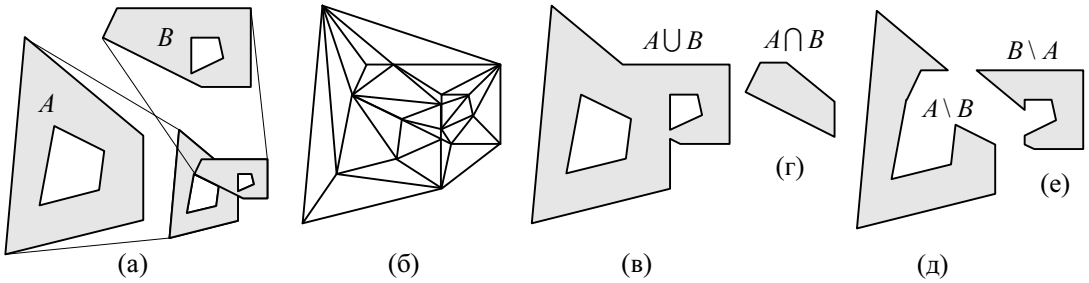

Рис. 6. Построение оверлеев: (а) — исходные многоугольники, (б) — триангуляция с ограничениями, (в) — объединение, (г) — пересечение, (д) и (е) — разности многоугольников

В общем виде алгоритм можно записать следующим образом.

Шаг 1. Стороны исходных многоугольников P_1 и P_2 (рис. 6 (а)) передается в качестве структурных линий в алгоритм построения триангуляции с ограничениями (рис. 6 (б)).

Шаг 2. Каждый треугольник триангуляции классифицируется по признаку попадания внутрь P_1 и P_2.

Шаг 3. Каждый треугольник T_i полученной триангуляции классифицируется в зависимости от выполняемой операции.

Вариант 1 (*объединение*): если $T_i \in P_1$ или $T_i \in P_2$, то $c_i := 1$, иначе $c_i := 0$.

Вариант 2 (*пересечение*): если $T_i \in P_1$ и $T_i \in P_2$, то $c_i := 1$, иначе $c_i := 0$.

Вариант 3 (*разность*): если $T_i \in P_1$ и $T_i \notin P_2$, то $c_i := 1$, иначе $c_i := 0$.

Шаг 4. Все треугольники, имеющие $c_i = 1$, объединяются в один многоугольник, который выбирается в качестве результата (рис. 6 (в)–(е)). *Конец алгоритма.*

Для выполнения первого шага алгоритма можно воспользоваться любым алгоритмом построения триангуляции с ограничениями. Обзор таких алгоритмов приведен, например, в [3].

В целом трудоемкость первого шага составляет в худшем случае $O(n\log(n_1+n_2))$, а в среднем — $O(n)$. Трудоемкость третьего шага, очевидно, является линейной — $O(n)$. Трудоемкость второго и четвертого шагов, как будет показано ниже, составляет также $O(n)$.

Таким образом, в целом алгоритм построения оверлеев на основе триангуляции имеет в худшем случае трудоемкость $O(n\log(n_1+n_2))$, а в среднем — $O(n)$.

3.1. Классификация треугольников. Рассмотрим второй шаг алгоритма построения оверлеев — задачу классификации треугольников триангуляции по признаку их попадания внутрь заданных многоугольников.

В простейшем случае можно для каждого отдельно взятого треугольника взять любую точку внутри него и проверить ее на попадание в заданный многоугольник. Однако тогда трудоемкость такого алгоритма станет квадратичной.

Ранее автором предлагался алгоритм, имеющий трудоемкость $O(nm_2)$, где m_2 — количество кон-

туров во втором многоугольнике [4]. В настоящей работе предлагается более эффективный алгоритм с трудоемкостью $O(n)$.

Итак, пусть для каждого треугольника необходимо выставить признак $c_i^j = 1$, если он находится внутри j-го многоугольника, и $c_i^j = 0$ в противном случае ($j \in \{1, 2\}$). Будем считать, что при вставке в триангуляцию структурных линий, принадлежащих границам многоугольников, для каждого фиксированного ребра отмечалось, к какому многоугольнику он относится. При этом возможно, что одно и то же фиксированное ребро может относиться к нескольким многоугольникам одновременно. Кроме того, если граница некоторого многоугольника проходит через какое-то ребро многократно, то это количество прохождений также должно быть отмечено. При рассмотрении попадания треугольников в многоугольник мы будем игнорировать ребра с четным количеством прохождений в соответствии с определением многоугольника.

Предлагаемый ниже алгоритм можно представить как последовательное "обкусывание" треугольников триангуляции (начиная с границы) до тех пор, пока не будут достигнуты фиксированные ребра. На первом этапе нужно "откусить" все внешние треугольники (рис. 7 (а)), на втором — внутренние (рис. 7 (б)), потом опять внешние (рис. 7 (в)) и т.д. до тех пор, пока не останется ни одного необработанного треугольника.

Приводимый алгоритм необходимо вызвать дважды: вначале для классификации треугольников по первому многоугольнику, а затем — по второму.

Алгоритм определения попадания треугольников в заданный многоугольник. Пусть задана триангуляция и для каждого фиксированного ребра отмечено, сколько раз через него проходит граница данного многоугольника.

Шаг 1. Каждый треугольник помечается как необработанный: $t_i := 0$.

Шаг 2. Помечаются метками $r_i := 1$ все фиксированные ребра, по которым граница региона проходит нечетное число раз. Остальные ребра помечаются метками $r_i := 0$.

Шаг 3. Устанавливается текущий режим "обкусывания" треугольников: $C := 0$ (поиск внешних треугольников).

Шаг 4. "Обкусываются" все внешние по отношению к заданному многоугольнику треугольники. Для этого выполняется проход вдоль границы триангуляции и при переходе через каждое ребро с $r_i = 0$ выполняется шаг A (начиная с соответствующего граничного треугольника; рис. 7 (а)).

Шаг 5. Если в триангуляции уже все треугольники являются "обкусанными" (т.е. для всех треугольников $t_i = 1$), то работа алгоритма завершается. Текущий режим "обкусывания" треугольников меняется на противоположный: $C := 1 - C$.

Шаг 6. Выполняется проход вдоль текущей границы "необкусанной" триангуляции и при переходе через каждое ребро с $r_i = 1$ на треугольник с $t_i = 0$ выполняется шаг A (начиная с этого треугольника; рис. 7 (б)). Переход на шаг 5.

Шаг A. Выполняется поиск всех треугольников в заданной замкнутой области алгоритмом типа растровой заливки с затравкой [2]. При этом границей области заливки являются ребра $r_i = 1$. Каждый найденный треугольник помечается метками $t_i := 1$, $c_j := C$ (рис. 7 (а)–(в)). Возврат в точку вызова этого шага. *Конец алгоритма.*

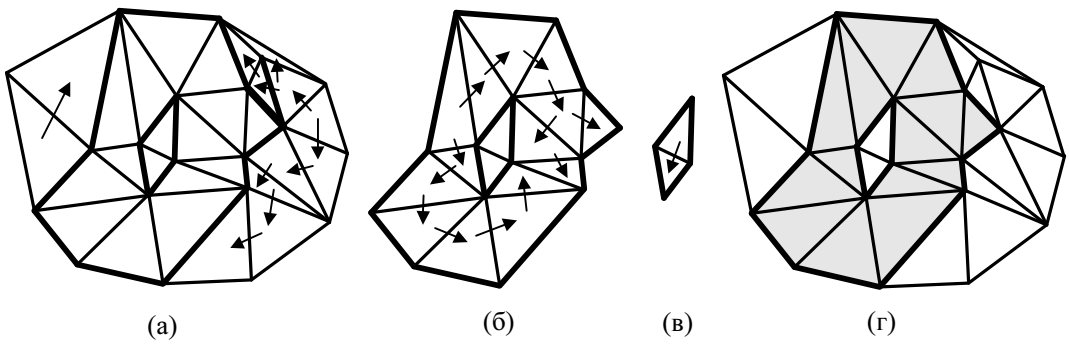

Рис. 7. Классификация треугольников: (а) — удаление внешних к многоугольнику треугольников; (б) — поиск внутренних треугольников, их маркировка и удаление; (в) — поиск внешних треугольников; (г) — найденные треугольники

Трудоемкость первых четырех шагов данного алгоритма составляет в сумме $O(n)$. Найдем сложность оставшихся шагов. На шаге А каждый треугольник триангуляции может быть пройден не более одного раза, т.к. алгоритм заливки с затравкой выполняется только начиная с треугольников $t_i = 0$, а каждый пройденный треугольник помечается меткой $t_i := 1$. Цикл на шагах 5 и 6 выполняется для ребер текущей границы триангуляции. При этом на шаге 6 выполняется вызов шага А для обнаружения всех треугольников, начиная с треугольников, смежных с текущей границей. Поэтому в результате работы шага 6 текущая граница полностью заменяется другими ребрами. Таким образом, на шаге 6 каждое ребро триангуляции будет пройдено не более одного раза. Это означает, что трудоемкость цикла на шагах 5 и 6 является линейной относительно размера триангуляции, т.е. составляет $O(n)$.

3.2. Выделение многоугольников из триангуляции. Теперь рассмотрим четвертый шаг алгоритма построения оверлеев — задачу выделения многоугольников из триангуляции. Она является обратной по отношению к предыдущей — задаче классификации. Нам нужно объединить треугольники, имеющие $c_i = 1$, в один многоугольник (рис. 8). Для этого можно использовать следующий алгоритм.

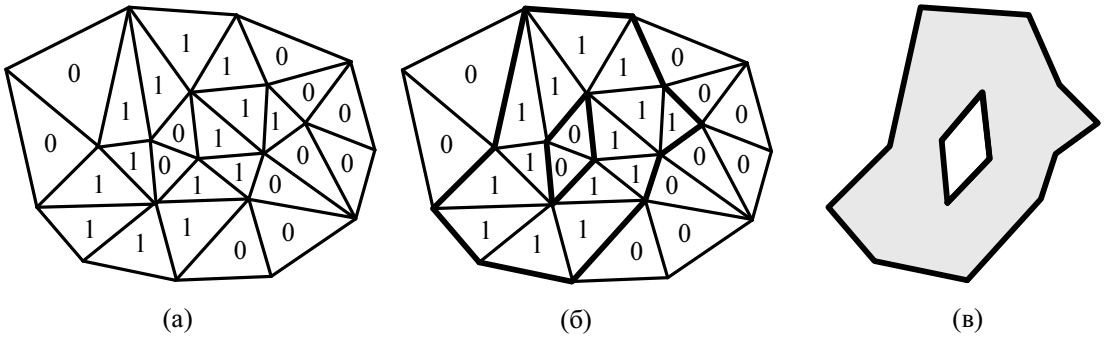

Рис. 8. Выделение многоугольника из триангуляции: (а) — исходные классифицированные треугольники, (б) — выделение ребер, (в) — объединение ребер в контуры многоугольника

Алгоритм выделения многоугольника из триангуляции.

Шаг 1. Очистить список готовых контуров многоугольника $K = \emptyset$.

Шаг 2. Найти все ребра, которые войдут в результирующий многоугольник, т.е. все ребра, у которых один смежный треугольник имеет $c_i = 1$, а другой смежный треугольник либо не существует, либо имеет $c_j = 0$. Найденные ребра пометить $r_i := 1$, а остальные ребра — $r_i := 0$ (рис. 8 (б)).

Шаг 3. Выполняется цикл по всем узлам v_i триангуляции. Пока из v_i выходит хотя бы одно ребро с $r_i = 1$, выполняется шаг А с этим узлом v_i. По окончании цикла работа алгоритма завершается.

Шаг А. Начиная с узла v_i, выполняется последовательный проход по ребрам триангуляции, имеющим $r_i = 1$, до тех пор, пока не будет достигнут начальный узел v_i. При этом ребра помечаются метками $r_i := 0$, чтобы исключить их повторное прохождение. Из пройденных ребер формируется новый контур и помещается в K. *Конец алгоритма.*

Трудоемкость работы рассмотренного алгоритма является линейной относительно размера триангуляции — $O(n)$, т.к. каждое ребро триангуляции будет пройдено не более одного раза.

Отметим, что приведенный алгоритм может строить самопересекающиеся контуры (типа "восьмерки"). Для устранения этого на шаге А нужно добавить дополнительную проверку, а именно: при движении через узлы при наличии нескольких альтернатив дальнейшего движения нужно выбирать очередное ребро, позволяющее повернуть максимально вправо относительно текущего направления движения. Но при этом трудоемкость алгоритма в худшем случае может возрасти до квадратичной, т.к. необходимо при выборе очередного направления движения выбирать из нескольких альтернатив. Тем не менее, т.к. в триангуляции каждый узел имеет в среднем только шесть исходящих ребер, то в среднем такой модифицированный алгоритм также завершит работу за время $O(n)$.

4. Заключение. Представленный в данной работе алгоритм построения оверлеев имеет бóльшую область определения, чем другие известные автору алгоритмы. По трудоемкости работы он также имеет лучшие характеристики, чем все остальные алгоритмы, за исключением некоторых алгоритмов, работающих только с выпуклыми многоугольниками (например, алгоритм О'Рурка). Важным достоинством предложенного алгоритма является высокая вычислительная устойчивость, что определяется, в первую очередь, используемым алгоритмом построения триангуляции с ограничениями. В табл. 1 приведены

сравнительные характеристики различных алгоритмов построения оверлеев.

Таблица 1.

Сравнительные характеристики различных алгоритмов построения оверлеев: n_1 — количество вершин первого многоугольника, n_2 — второго, n_0 — количество новых точек пересечения сторон двух многоугольников, $n = n_1 + n_2 + n_0$

Алгоритм	Тип исходных многоугольников	Многоконтурность	Устойчивость	Трудоемкость
О'Рурка	Выпуклые	—	—	$O(n)$
Сазерленда–Ходжмана (только пересечение)	Один выпуклый, другой общий	+	+	$O(n_1 n_2 \log n_2)$
Вейлера–Азертона	Простые с ориентированными контурами	+	—	$O(n \log(n_1 + n_2))$
Маргалита–Кнотта	Вершинно-полные	—	—	$O(n^2)$
Маргалита–Кнотта, теоретическое улучшение	Вершинно-полные	—	—	$O(n \log n)$
Триангуляционный	Общие	+	+	$O(n \log(n_1+n_2))$ — в худшем, $O(n)$ — в среднем

Автором была выполнена реализация некоторых из приведенных в таблице алгоритмов (Сазерленда–Ходжмана, Вейлера–Азертона, Маргалита–Кнотта и предложенного в данной работе триангуляционного) и проведено их сравнение на больших наборах реальных данных, встречающихся в геоинформационных системах.

В целом можно сказать, что в связи с достаточно высокой алгоритмической сложностью предложенного алгоритма, на простых видах исходных данных с небольшим количеством вершин он зачастую уступает по скорости другим алгоритмам. Однако на сложных многоугольниках с большим количеством вершин в исходных многоугольниках он существенно превосходит все остальные алгоритмы, что, безусловно, подтверждается полученной линейной асимптотической оценкой трудоемкости в среднем.

СПИСОК ЛИТЕРАТУРЫ

1. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение. М.: Мир, 1989.
2. *Роджерс Д., Адамс Дж.* Математические основы машинной графики. М.: Машиностроение, 1980.
3. *Скворцов А.В.* Алгоритмы построения триангуляции с ограничениями // Вычислительные методы и программирование. 2002. **3**. 82–92 (http://num-meth.srcc.msu.su).
4. *Скворцов А.В., Костюк Ю.Л.* Применение триангуляции для решения задач вычислительной геометрии // Геоинформатика: Теория и практика. Вып. 1. Томск: Изд-во Томск. ун-та, 1998. 127–138.
5. *Margalit A., Knott G.D.* An algorithm for computing the union, intersection of difference of two polygons // Computers & Graphics. 1989. **13**, N 2. 167–183.
6. *O'Rourke J., Chien C.-B., Olson T., Naddor D.* A new linear algorithm for intersecting convex polygons // Computer Graphics and Image Processing. 1982. **19**. 384–391.
7. *Sutherland I.E., Hodgman G.W.* Reentrant polygon clipping // CACM. 1983. **26**. 868–877.
8. *Weiler K., Atherton P.* Hidden surface removal using polygon area sorting // Computer Graphics. 1977. **11**. 214–222.

Поступила в редакцию
10.03.2002