УДК 519.688

О.В. Ченцов, А.В. Скворцов

ОБЗОР АЛГОРИТМОВ ПОСТРОЕНИЯ ОВЕРЛЕЕВ МНОГОУГОЛЬНИКОВ

Проводится обзор алгоритмов пересечения, объединения и разности многоугольников и сравнение их по ряду таких параметров, как вычислительная устойчивость, трудоемкость, скорость работы, простота программной реализации. В работе описаны как широко известные, так и менее распространенные алгоритмы, в том числе и не опубликованные никем ранее, но доступные через интернет.

Во многих приложениях вычислительной геометрии и компьютерной графики, в системах автоматизированного проектирования (САПР), в геоинформационных системах (ГИС), в различных векторных графических редакторах основные объекты описываются с помощью плоских многоугольников.

Одной из основных операций над многоугольниками является операция построения оверлеев (объединения, пересечения или разности).

Несмотря на внешнюю простоту постановки, многие существующие алгоритмы её решения обладают существенными недостатками для использования на практике. Основная проблема связана с потерей вычислительной точности при вычислениях, что приводит к некорректной работе алгоритмов. Во многих алгоритмах устранение такого рода проблем с точностью приводит к существенному усложнению реализации алгоритма, что не всегда отражается в оригинальных работах авторов.

С другой стороны, существует ряд алгоритмов, которые в настоящее время широко не известны по ряду причин: либо из-за того, что они напечатаны в недоступных источниках, либо они не опубликованы вообще. Такая ситуация сложилась с некоторыми алгоритмами, на основе которых разработаны некоторые общедоступные библиотеки программ построения оверлеев. Такие библиотеки доступны в интернете, доступны их общие описания, однако зачастую научных статей с описанием принципов работы алгоритмов и их сравнения с аналогами нет.

В настоящей работе предпринята попытка обобщения результатов исследований различных алгоритмов построения оверлеев.

ПОСТАНОВКА ЗАДАЧИ

Рассмотрим евклидово пространство размерности 2 с правой декартовой системой координат. Будем считать, что многоугольники, подающиеся на вход алгоритмов построения оверлеев, расположены в этой плоскости.

Определение. *Ребром* называется направленный отрезок с началом в точке a и концом в точке b, не совпадающей с a, который обозначается как $E(a,b)$. При этом будем говорить, что для точки b ребро E является *входящим*, а для точки a – *выходящим*.

Определение. *Контуром* называется множество ребер $\{E_0,E_1,...,E_{n\text{-}1}\}$ такое, что $n>2$ и $\forall_i \in\{0,...,n-1\}$: $E_{i-1}=E(v_{i-1},v_i)$, $E_i=E(v_i,v_{i+1})$. Точка v_i, для которой ребра E_{i-1} и E_i являются соответственно входящим и выходящим, будем называть i-й вершиной контура C, ребра E_{i-1} и E_i – соседними или смежными. Порядок обхода ребер контура $C=\{E_0,E_1,...,E_{n\text{-}1}\}$ с увеличением индекса $i\to i+1$ будем называть *прямым* направлением обхода, а противоположный ему – *обратным*. Заметим, что одной точке плоскости может соответствовать несколько вершин контура, такие вершины будем называть *кратными*.

Определение. *Областью* называется ограниченное полигональное открытое связное множество на евклидовой плоскости, определенное с точностью до множества меры нуль. Под *полигональностью* понимается то, что граница области состоит из замкнутых ломаных.

Рассмотрим границу dA некоторой области A. Если ориентация dA согласована с ориентацией A, мы приходим к соответствию между границей области и набором контуров на плоскости. Для того, чтобы оно было взаимно-однозначным, добавим к традиционному определению два ограничения. Итак, рассмотрим, контур C, содержащий n ребер.

Определение. Контур C называется *ограничивающим* контуром области A, если выполняются следующие условия:

1) $C\subset dA$;

2) обход C в прямом направлении является положительным обходом, т.е. обходом, при котором область A остается слева;

3) $\forall_i \in\{0,...,n-1\}:\exists\ \varepsilon>0:\forall\ x\in\varepsilon^+(\upsilon_i):x\in A$;

4) $\forall_i \in\{0,...,n-1\}:\exists\ \varepsilon>0:\forall\ x\in\varepsilon^-(\upsilon_i):x\notin A$.

Определение. *Общим* полигоном или многоугольником называется множество всех ограничивающих контуров некоторой области.

Внешним контуром области A назовем ограничивающий ее контур, положительный обход которого производится против часовой стрелки. *Внутренним* контуром области A назовем ограничивающий ее контур, положительный обход которого производится по часовой стрелке. Очевидно, что в произвольном полигоне может быть один и только один внешний контур.

Определение. Многоугольник называется *выпуклым*, если определяемое им множество точек является выпуклым.

Определение. Многоугольник называется *простым*, если он имеет, по крайней мере, две различных точки, если ни одна пара непоследовательных ребер не разделяет точку и если каждая пара последовательных ребер разделяет только одну точку.

Определение. Многоугольник называется *регулярным*, если он не имеет пересекающихся или «висящих» ребер и совпадающих точек.

Определение. *Вершинно-полным* называется многоугольник, который не имеет пересекающихся ребер, однако у него могут быть совпадающие вершины и коллинеарные ребра.

Определение. *Ориентированным* называется многоугольник, любые два ребра которого либо не пересекаются, либо коллинеарны, либо пересекаются в точке, являющейся конечной точкой как минимум одного из ребер.

Указанные классы полигонов формируют следующую иерархию: *выпуклые* $\subset$ *простые* $\subset$ *регулярные* $\subset$ *вершинно-полные* $\subset$ *ориентированные* $\subset$ *общие*.

Итак, рассмотрим два двумерных многоугольника A и B. Под выполнением некоторой булевой операции над многоугольниками A и B понимается выполнение операции над описываемыми ими областями и получение множества многоугольников, описывающего полученную область.

Определим следующий набор возможных операций:

1. Объединение (рис. 1,*а*):

$A \cup B = \{(x,y) : (x,y) \in A \text{ или } (x,y) \in B\}$.

2. Пересечение (рис. 1,*б*):

$A \cap B = \{(x,y) : (x,y) \in A \text{ и } (x,y) \in B\}$.

3. Разность (рис. 1,*в*):

$A - B = \text{clos } \{(x,y) : (x,y) \in A \text{ и } (x,y) \notin B\}$.

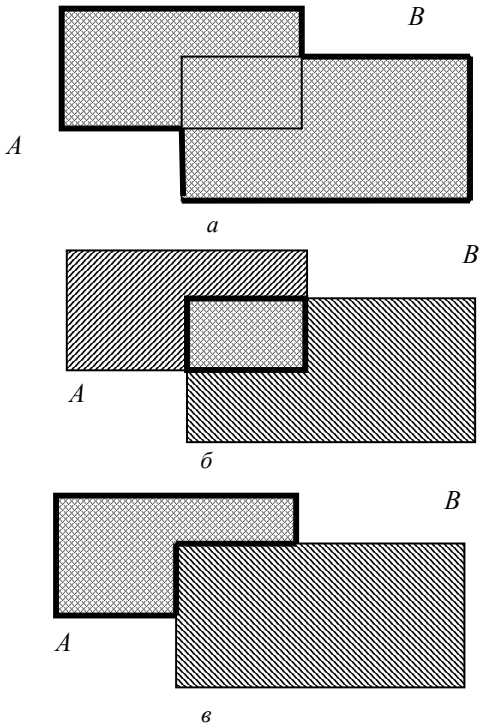

Рис. 1. Набор операций над многоугольниками A и B

АЛГОРИТМ САЗЕРЛЕНДА – ХОДЖМАНА

Алгоритм Сазерленда – Ходжмана является одним из самых первых и простых алгоритмов отсечения многоугольников [8]. Он был базовым алгоритмом для процессора Кларка [9], который был предтечей алгоритмов, заложенных в классических программах первых компьютеров Silicon Graphics Inc. Этот алгоритм рассекает исходный *общий* многоугольник любым выпуклым многоугольником. Отсекающий многоугольник обычно называют *отсекающим окном*. Суть алгоритма заключается в том, что отсекаемый многоугольник последовательно отсекается каждой границей отсекающего окна (рис. 2).

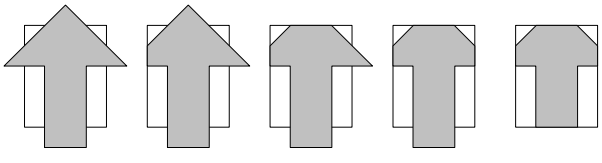

Рис. 2. Отсечение многоугольника выпуклым отсекателем в алгоритме Сазерленда – Ходжмана

Алгоритм обрабатывает пересечение каждого ребра исходного многоугольника с каждым ребром отсекателя, сохраняя вершины, находящиеся внутри ребра и точки пересечения ребер. Далее алгоритм повторяется, на вход подаются полученный на предыдущем шаге временный многоугольник и другое ребро отсекателя. Существует четыре случая при обработке возможного пересечения ребра отсекаемого многоугольника и ребра отсекателя (допустим, что ребро отсекаемого многоугольника идет от вершины S к вершине P):

1. Обе вершины (S и P) ребра отсекаемого многоугольника внутри ребра отсекателя, вершина P добавляется в выходной список (рис. 3, *а*).

2. Если вершина S внутри ребра отсекателя, а вершина P снаружи его, то в выходной список заносится точка пересечения i (рис. 3, *б*).

3. Если обе точки (S и P) ребра снаружи ребра отсекателя, тогда в выходной список не попадает ничего (рис. 3, *в*).

4. Если вершина P внутри ребра отсекателя, а вершина S снаружи его, то в выходной список заносится точка пересечения i и вершина P (рис. 3, *г*).

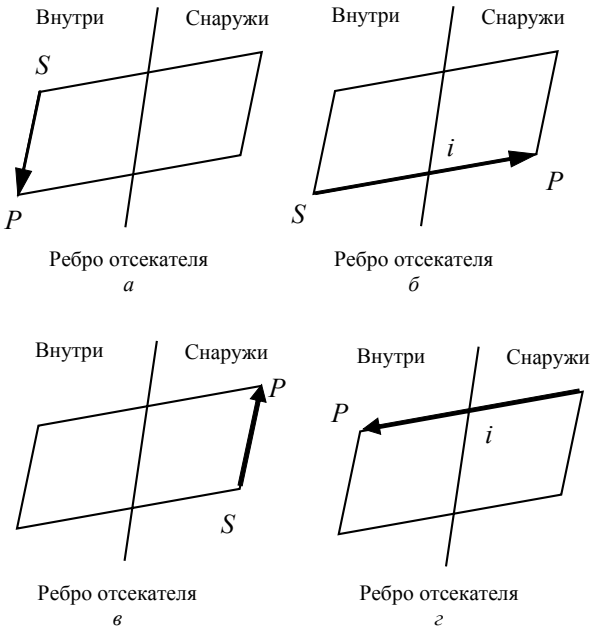

Рис. 3. Варианты пересечения ребра исходного многоугольника и ребра отсекателя в алгоритме Сазерленда – Ходжмана

В завершении работы алгоритма необходимо рассмотреть первую вершину и последнее ребро отсекаемого многоугольника. Если первая вершина находится внутри текущего ребра, мы сохраняем ее в списке вершин временного выходного многоугольника, иначе мы отбрасываем ее. Что касается последнего ребра, следует заметить, что если в нашем выходном временном многоугольнике пусто, то весь исходный многоугольник не должен быть виден в отсекающем многоугольнике, т.е. мы можем прервать работу. В ином случае, если последнее ребро отсекаемого многоугольника пересекается с ребром отсекателя, то точка пересечения должна быть добавлена во временный выходной многоугольник.

К «плюсам» алгоритма стоит отнести его простоту, малую требовательность к памяти (всего два списка цепочек), к «минусам» – довольно высокую трудоемкость (O ($n_1 n_2 \log n_2$)), пригодность только для выпуклых отсекателей, невозможность обобщения для построения объединения и разности многоугольников. Кроме этого, к недостаткам алгоритма можно отнести и то, что в результате могут образоваться вырожденные многоугольники, когда некоторые рёбра могут налагаться друг на друга, образуя «перешейки» нулевой ширины.

АЛГОРИТМ О'РУРКА

Один из наиболее простых и элегантных алгоритмов пересечения выпуклых многоугольников был предложен О'Рурком [11]. Он базируется на более простом и грубом методе, описанном в [1].

Основная идея алгоритма О'Рурка состоит в следующем. Допустим, даны два выпуклых многоугольника P с L вершинами и Q с M вершинами. 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. Для каждого многоугольника объявляются текущие вершины p_i и q_i, кроме этого, текущие ребра оканчиваются вершинами p_i и q_i. Идея заключается в том, чтобы не двигаться по той границе (P или Q), текущее ребро которой еще может содержать необнаруженную точку пересечения. Существует четыре возможных варианта взаимного расположения вершин p_i и q_j и ребер $\overline{p_{i-1}p_i}$ и $\overline{q_{j-1}q_j}$ (остальные ситуации сводятся к указанным четырем путем замены ролей P и Q). В одном из этих вариантов (рис. 4, *а*) мы продвинемся по P, так как текущее ребро $\overline{q_{j-1}q_j}$ из Q может содержать необнаруженную точку пересечения; по тем же причинам в случае (рис. 4, *б*) мы продвинемся по границе Q.

Если все пересечения на текущем ребре из P уже обнаружены, в то время как текущее ребро из Q все еще может содержать необнаруженное пересечение, мы продвигаемся вдоль P (рис. 4, *в*); кроме этого, здесь мы должны вычислить точку пересечения ребер $\overline{p_{i-1}p_i}$ и $\overline{q_{j-1}q_j}$. В последнем случае (рис. 4, *г*) выбор произволен и можно выбрать, например, Q.

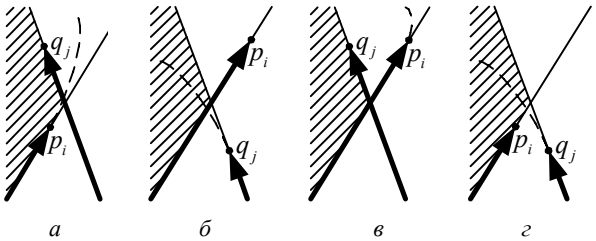

Рис. 4. Варианты взаимного расположения текущих вершин и ребер в алгоритме О'Рурка

В целом, можно сказать, что алгоритм О'Рурка эффективен и прост в реализации, кроме этого, он может быть легко модифицирован для построения объединения и разности многоугольников.

АЛГОРИТМ ВЕЙЛЕРА – АЗЕРТОНА

Алгоритм Вейлера – Азертона [4, 5] является гораздо более мощным алгоритмом отсечения многоугольников по сравнению с ранее перечисленными. Он пригоден для обработки областей с «дырами» (без самопересечений) и для любых регулярных отсекателей, однако, достигается это ценой заметно большей сложности и меньшей скорости работы.

Отсекаемый многоугольник называется объектом (SP), а отсекающий многоугольник – отсекателем (CP). В процессе пересечения не создаются новые ребра, поэтому число выходных многоугольников минимизировано.

Алгоритм описывает SP и CP как циклический список вершин. Внешние границы многоугольников представляются упорядоченными по часовой стрелке, а внутренние границы и «дыры» упорядоченными против часовой стрелки. Отсюда, при перемещении по списку вершин, внутренняя область многоугольника должна всегда находиться справа. Границы SP или CP могут пересекаться, а могут не пересекаться. Если они пересекаются, пересечения возникают дважды. Одно из пересечений происходит, когда ребро SP входит внутрь CP, а второе, когда выходит. По существу, алгоритм начинается на «входящем» пересечении и следует по часовой стрелке до внешней границы SP до тех пор, пока не будет найдено пересечение с CP. На пересечении делается правый поворот и выполняется обход по часовой стрелке внешней области CP, пока не будет найдено пересечение с SP. Далее, снова на пересечении делается правый поворот и выполняется обход по SP. Процесс продолжается до тех пор, пока не будет достигнута стартовая точка. Внутренние границы SP обходятся против часовой стрелки.

Ниже описаны основные шаги алгоритма:

Шаг 1. Нахождение точек пересечения SP и CP – каждое пересечение добавляется в список вершин SP и CP. При этом касания не считаются пересечением, т.е. когда вершина или ребро отсекаемого многоугольника инцидентна или совпадает со стороной отсекателя (рис. 5 и 6). Каждая пересекающаяся вершина помечается и устанавливается двунаправленная связь между списками SP и CP.

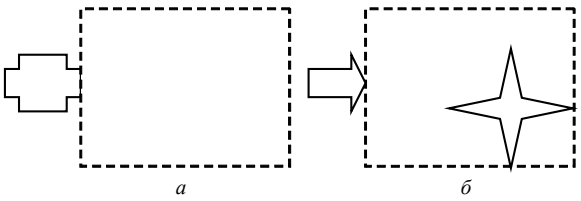

Рис. 5. Случаи, не считающиеся пересечением, в алгоритме Вейлера–Азертона

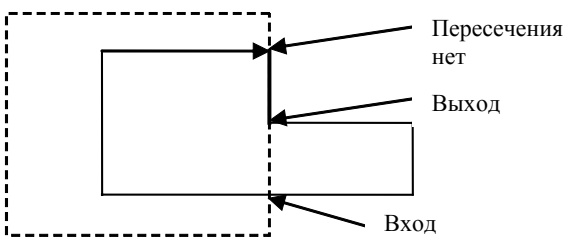

Рис. 6. Частный случай пересечения многоугольников в алгоритме Вейлера – Азертона

Шаг 2. Обработка непересекающихся границ многоугольников – вводятся два списка: один для границ, находящихся внутри CP, второй – для находящихся снаружи. Границы CP, находящиеся снаружи SP, игнорируются. Границы CP внутри SP формируют «дыры» SP. Поэтому копия границ CP идет в оба списка – «внутренний» и «наружный». Границы помещаются в соответствующий список.

Шаг 3. Создание двух списков точек пересечений – первый, «входящий» список, содержит только точки пересечения, когда ребро SP входит внутрь CP. Другой, «выходящий» список, содержит точки пересечения, когда ребро SP выходит изнутри CP. Тип пересечения будет чередоваться вдоль границы. Таким образом, только одно определение необходимо для каждой пары пересечений.

Шаг 4. Выполнение пересечения. Многоугольники внутри CP находятся, используя следующую процедуру:

Шаг 4.1. Точка пересечения удаляется из «входящего» списка. Если список пуст, тогда процесс завершен.

Шаг 4.2. Выполняется обход списка вершин SP, пока не будет найдена точка пересечения. Список SP вплоть до найденной точки копируется во «внутренний» список.

Шаг 4.3. Используя ссылку, переходим на список вершин CP.

Шаг 4.4. Выполняется обход списка вершин CP, пока не будет найдена точка пересечения. Список CP вплоть до найденной точки копируется во «внутренний» список.

Шаг 4.5. Возвращается обратно на список вершин SP.

Шаг 4.6. Повторяем до тех пор, пока не будет достигнута стартовая точка. На этой точке заканчивается новый внутренний многоугольник. Конец алгоритма.

Многоугольники снаружи CP находятся, используя ту же процедуру, за исключением того, что начальная точка пересечения получается из «выходящего» списка и список вершин CP обрабатывается в обратном направлении. Списки многоугольников копируются в «наружный» список.

Более подробное описание алгоритма может быть найдено в [2, 3].

Трудоемкость вычислений пересечений равна $O(n_1,n_2)$, где n_1 и n_2 – соответственно количество вершин SP и CP. При применении методов пространственного индексирования рёбер исходных многоугольников, например с помощью *R*-деревьев, трудоемкость может быть снижена до $O(n_1+n_2)$.

Такие минусы, как высокая сложность при отработке большого количества особых случаев [2], возникающих при реализации, и невысокая скорость работы не позволяют рекомендовать данный алгоритм для практического применения.

АЛГОРИТМ ЛЕОНОВА

Практически все алгоритмы вычисления пересечения, объединения или разности многоугольников реализуют набор операций, не являющийся замкнутым, так как в результате их выполнения могут получаться контуры с самокасаниями, что неприемлемо в качестве исходных данных для всех алгоритмов, и с отверстиями, которые, например, не допускаются алгоритмом Шутте [6]. Кроме этого, многие из широко известных алгоритмов не позволяют корректно обрабатывать вершины, в которых сходится более двух ребер многоугольников.

Алгоритм Леонова [7] лишен этих недостатков. В качестве исходных данных здесь могут быть произвольные полигональные многосвязные области с отверстиями и неограниченной кратностью вершин. Результат, полученный с помощью алгоритма, удовлетворяет его же предусловию.

Алгоритм состоит из четырех этапов:

Этап 1. Обработка пересечений ребер.

Этап 2. Маркировка контуров и ребер.

Этап 3. Получение результирующих контуров.

Этап 4. Создание результирующего региона.

На *первом* этапе находятся все пары несмежных пересекающихся ребер регионов *A* и *B*. Если пересечение ребер не точка, а отрезок, то концы этого отрезка считаются двумя точками пересечения. Если точки пересечения не совпадают с концевыми точками ребер, в исходные регионы добавляются новые вершины с координатами соответствующих точек. Следует отметить, что добавление точек пересечения не изменяет области, описываемые регионами *A* и *B*. Вершины, соответствующие точкам пересечения ребер, называются вершинами-пересечениями.

После выполнения первого этапа алгоритма выполняются следующие условия:

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

2. Произвольное ребро контура *C* может либо геометрически совпадать с одним из ребер *M* (такие ребра будем называть сопряженными), либо находиться целиком (за исключением, может быть, концевых точек) внутри или вне области, описываемой *M*.

3. Если контур *C* не содержит вершин-пересечений, то он лежит целиком внутри или вне области, описываемой *M*.

На *втором* этапе рассматривается некоторый ограничивающий контур *C* одного из регионов *A* и *B*. За *M* обозначается регион, к которому не принадлежит *C*. На предыдущем этапе все точки пересечения были добавлены в исходные регионы.

Пометка контура *C* или ребра, принадлежащего контуру *C*, делается в зависимости от геометрического положения относительно *M*.

Сборка результата осуществляется для всех контуров единообразно. На *третьем* этапе 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* этапе происходит формирование результирующего контура *R*. Задача правильного формирования *R* очень проста, так как полученные на предыдущем этапе внешние и внутренние контуры имеют правильную ориентацию. Для каждого внешнего контура создается отдельный полигон, внешним контуром которого он и будет являться. Внутренние контуры на предыдущем этапе сохраняются во вре-

менном списке, а затем помещаются в результирующие полигоны. Найти нужный полигон несложно, для этого достаточно определить минимальный из внешних контуров, содержащих данный внутренний контур. Регион R будет состоять из множества полученных таким образом полигонов.

Вычислительная трудоемкость алгоритма различается на разных этапах. В среднем она равна

$$O((n_1+n_2)\log(n_1+n_2)+n_0+z\log(n_1+n_2)),$$

где (n_1+n_2) – общее число вершин регионов A и B; z – общее число контуров регионов A и B; n_0 – максимальное число новых вершин.

Исходя из вышеперечисленного, можно сделать вывод, что алгоритм Леонова является одним из лучших по важнейшим из параметров (скорости, затратам на память, численной устойчивости, обработке вырожденных ситуаций).

АЛГОРИТМ ШУТТЕ

Алгоритм Шутте [6] базируется на алгоритме Вейлера – Азертона [4], 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 $A\cap B$, $A\backslash B$, A/B. *Конец алгоритма*.

На *первом* этапе точки пересечения добавляются к спискам вершин многоугольников. Новые вершины одного многоугольника содержат ссылку на соответствующие вершины другого многоугольника. Получившиеся многоугольники называются расширенными.

Следует заметить, что возможен вариант, когда не будет найдено ни одной точки пересечения. В данном случае возможна одна из следующих ситуаций:

1. Многоугольник A внутри многоугольника B. Здесь любая вершина из A находится внутри B, что легко может быть проверено. Если A внутри B, мы произвольно разбиваем B на два многоугольника – A будет находиться где-то на пересечении. Мы делаем разбиение, так как не хотим появления многоугольников с «дырами».
2. Многоугольник B находится внутри многоугольника A. Аналогично случаю, описанному выше.
3. Многоугольники A и B разделены.

Идея, заложенная на *втором* этапе алгоритма, заключается в том, что если одна из вершин ребра соединена с другим многоугольником, можно проверить, располагается ли она внутри или снаружи второго многоугольника. Если ни одна из вершин ребра не лежит на ребре второго многоугольника, то для проверки, находится ли исходное ребро внутри, можно использовать обе вершины.

На *третьем* этапе все ребра обоих расширенных многоугольников дублируются в прямые и обратные ребра, которые называются направленными. Разделенные ребра дублируются только раз. Для каждого направленного ребра мы ищем минимальный, упорядоченный по часовой стрелке многоугольник, частью которого данное ребро является. Поиск начинается по всем вершинам обоих многоугольников по двум направлениям. Повторное добавление одинаковых многоугольников запрещается – каждое ребро включается только один раз для каждого направления.

Для проведения классификации на четвертом этапе вводится термин «отцовство». «Отцовство» определяется как отношения между ребрами и исходными многоугольниками и между минимальными многоугольниками и исходными многоугольниками. Если «отцовство» доказано, значит объект является частью исходного многоугольника. Если «отцовство» ложно, значит объект не является частью исходного многоугольника. Кроме значений «доказано» и «ложно», «отцовство» может быть «неизвестным» и «смешанным». «Неизвестное» означает, что убедительных подтверждений нет. «Смешанное» означает, что существуют противоречащие факты. Это возможно только для минимальных многоугольников и означает, что минимальный многоугольник – это «дыра» между двумя исходными многоугольниками.

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.

Трудоемкость алгоритма при операциях нахождения пересечений и маркировки ребер равна $O(n_1n_2)$.

Можно сделать вывод, что, хотя Шутте модифицировал алгоритм Вейлера – Азертона более четким разделением этапов и новым алгоритмом маркировки ребер (edge labeling), его алгоритм обладает более существенными ограничениями на входные данные и не избавился от врожденных недостатков.

АЛГОРИТМ ХОЛВЕРДА

Входными данными *скан-линейного* алгоритма Холверда [12] являются два набора многоугольников. Один из наборов является первым операндом булевой операции, другой – вторым операндом. Основная идея состоит в том, чтобы разбить плоскость на секции, называемые *закрытыми областями*. Закрытые области – это области, которые могут быть описаны простыми многоугольниками (без самопересечений и самоперекрытий).

Результат булевой операции может быть получен, если мы сможем выяснить принадлежность каждой из

закрытых областей одному или обоим наборам многоугольников, для этого необходимо вычислить пересечения всех сегментов многоугольников.

Можно выделить следующие основные этапы алгоритма:

Шаг 1. Конвертация всех многоугольников в графы. Один многоугольник становится одним графом.

Шаг 2. Объединение всех графов в один большой граф. Такой граф идеален для скан-линейного алгоритма, так как ссылки графа могут быть отсортированы без потери структуры отображения. Только ссылки получат другой порядок в списке, позиция же узлов не изменится. Следует заметить, что каждая ссылка в графе получает уникальный номер, содержащий информацию принадлежности тому или иному многоугольнику.

Шаг 3. Вычисление точек пересечения между графами, используя скан-луч, и вставка их как дополнительных вершин или сегментов. *Метод скан-луча* используется несколько раз в этом алгоритме. Идея состоит в сканировании изображения вертикальным скан-лучом. *Скан-луч* – это пространство между двумя последовательными вертикальными скан-линиями. Скан-линия генерируется каждым узлом в графе. В первую очередь, все начальные узлы ссылок графа сортируются по минимальной X координате (если X одинаково, то по Y). Начальный узел одной ссылки является всегда конечным узлом другой ссылки просто потому, что все многоугольники являются замкнутыми петлями. То есть, когда мы генерируем скан-линию для каждого начального узла всех ссылок, можно быть уверенным, что все узлы графа будут пройдены.

Шаг 4. Установка соответствующих флагов сегментов одновременно для всех булевых операций, используя скан-луч.

Шаг 5. Получение для данной булевой операции результирующих многоугольников. В принципе, на данном этапе необходимо найти подходящую ссылку, помеченную для данной операции, и двигаться по связанным с первой ссылкам, которые также помечены для этой операции. Когда мы снова доходим до стартовой ссылки, можно извлекать один многоугольник, и так далее, пока не будут получены все многоугольники.

Данный этап выполняется за пять шагов:

Шаг 5.1. Уничтожение ненужных ссылок – удаление ссылок, для которых не установлен флаг участия в данной булевой операции.

Шаг 5.2. Выделение внутренних и внешних контуров (рис. 7) из общего графа и помещение их в список графов. На внутренние контуры ставится отметка «дыра».

Шаг 5.3. Создание одного графа – слияние всех контурных графов обратно в один граф (необходимо для связывания «дыр» с помощью скан-луча).

Шаг 5.4. Связывание «дыр» – использование сканлуча для связывания внутриконтурных графов с внешнеконтурными.

Шаг 5.5. Выделение всех связанных графов (внешний и внутренний контур как один граф) из общего графа.

Шаг 5.6. Конвертация графов обратно в многоугольники.

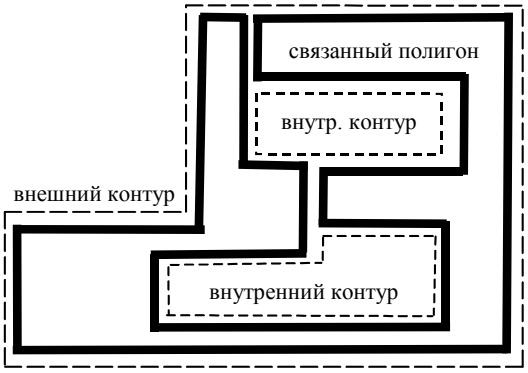

Рис. 7. Внутренние и внешние контуры многоугольника в алгоритме Холверда

Из недостатков алгоритма следует отметить то, что на входе нельзя использовать самопересекающиеся многоугольники. Однако оригинальность, высокая скорость работы, корректное округление координат точек пересечения и довольно высокая вычислительная устойчивость позволяют выделить его из общего ряда.

АЛГОРИТМ МАРГАЛИТА – КНОТТА

А. Маргалит и Г. Кнотт, авторы одноименного алгоритма, обратили внимание на то, что многие алгоритмы пересечения, объединения и разности многоугольников не очень практичны, не способны обрабатывать сложные случаи, сложны в реализации, и предложили свой алгоритм, ориентируясь, в первую очередь, на эффективность и простоту реализации [10]. Алгоритм делится на две основные стадии. Первая стадия – это классификация линейных сегментов входных многоугольников, а вторая – конструирование результирующих многоугольников.

В первую очередь, алгоритм классифицирует оригинальные вершины каждого многоугольника на нахождение внутри, снаружи или на границе другого многоугольника. Затем ищутся все точки пересечения между многоугольниками. Для каждого многоугольника оригинальные вершины вместе с точками пересечения помещаются в такую структуру данных, что две соседние точки определяют оригинальное ребро или часть оригинального ребра результирующего многоугольника (или, иначе говоря, реберный фрагмент).

Далее алгоритм 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. Эти три множества реберных фрагментов обозначаются как $F_p^{\text{in}}(Q)$, $F_p^{\text{on}}(Q)$ и $F_p^{\text{out}}(Q)$. Множество граничных реберных фрагментов $F_p^{\text{on}}(Q)$ в дальнейшем может быть разбито на две части, в зависимости от того, в ту же или противоположную сторону фрагментам P направлены фрагменты Q. Эти множества обозначаются как $F_p^{\text{on}\uparrow\uparrow}(Q)$ и $F_p^{\text{on}\uparrow\downarrow}(Q)$. Это тонкое деление позволяет нам создавать только регулярные результирующие многоугольники, для которых необходимо не

использовать реберные фрагменты из множества $F_p^{\text{on}\uparrow\downarrow}(Q)$.

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.

Этот алгоритм относительно прост и практичен. При реализации можно использовать хэш-таблицы и другие простые структуры данных. Элементарные манипуляции с этими структурами данных производительны, следовательно, минимизируется время и требуемый объем памяти. В алгоритме не нужно обрабатывать большое число специальных случаев, и поэтому он может легко работать с любой парой вершинно-полных многоугольников. Трудоемкость алгоритма составляет в худшем случае $O(n^2)$, однако на практике она может быть значительно улучшена в среднем до $O(n\log n)$.

ТРИАНГУЛЯЦИОННЫЙ АЛГОРИТМ

Основная идея алгоритма построения оверлеев многоугольников с помощью триангуляции [13] 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.

В общем виде алгоритм можно записать следующим образом (рис. 8):

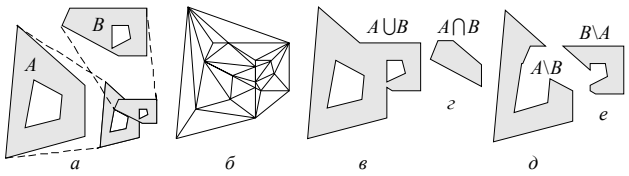

Рис. 8. Построение оверлеев: *а* – исходные многоугольники; *б* – триангуляция с ограничениями; *в* – объединение; *г* – пересечение; *д* и *е* – разности многоугольников

Шаг 1. Стороны исходных многоугольников A и B (рис. 8,*а*) передаются в качестве структурных линий в алгоритм построения триангуляции с ограничениями (рис. 8,*б*).

Шаг 2. Каждый треугольник триангуляции классифицируется по признаку попадания внутрь A и B.

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

Вариант 1 (объединение):

если $T_i \in A$ или $T_i \in B$, то $c_i := 1$, иначе $c_i := 0$.

Вариант 2 (пересечение):

если $T_i \in A$ и $T_i \in B$, то $c_i := 1$, иначе $c_i := 0$.

Вариант 3 (разность):

если $T_i \in A$ и $T_i \in B$, 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. 8,*в–е*). *Конец алгоритма.*

Для выполнения первого шага алгоритма можно воспользоваться любым алгоритмом построения триангуляции с ограничениями. Обзор таких алгоритмов приведен, например, в [14].

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 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 – $O(n)$.

В целом можно сказать, что в связи с достаточно высокой алгоритмической сложностью триангуляционного 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.

ЛИНЕЙНО-УЗЛОВОЙ АЛГОРИТМ

Основная идея рассматриваемого алгоритма [15] заключается в предварительном построении линейно-узловой модели – геометрического планарного графа специального вида, ребра которого должны соответствовать отрезкам исходных границ полигонов (происхождение вводимого автором термина «линейно-узловой» связано с используемыми в геоинформатике похожими топологическими моделями данных – покрытиями, называемыми также линейно-узловыми моделями).

Затем после построения графа производится классификация ребер графа по признаку вхождения в результирующий полигон. Для классификации всех ребер требуется умение выполнять две операции: 1) определять расположение полигона относительно ребра линейно-узловой модели и 2) определять, попадает ли некоторое ребро на границу, внутрь или вне полигона.

На последнем этапе алгоритма выполняется сборка отрезков графа в последовательность отрезков границы требуемого полигона. Для этого достаточно после классификации пометить входящие в результат ребра как непройденные, а остальные – как пройденные и запустить алгоритм выделения контуров.

Обратим внимание, что в результате работы предложенного алгоритма будет построен полигон, в котором контуры не будут взаимно- и самопересекаться,

и каждый контур будет задан в таком порядке обхода точек, что полигон всегда находится справа по ходу обхода.

Линейно-узловой алгоритм построения оверлеев позволяет явно учесть многочисленные тонкие эффекты, возникающие на пороге точности вычислений. Данный алгоритм имеет такой же порядок трудоемкости, что и алгоритм Маргалита – Кнотта [10], но в среднем в 1.5 раза медленнее последнего. Тем не менее предлагаемый алгоритм имеет несколько большую область определения и достаточно простую структуру для программной реализации.

ЗАКЛЮЧЕНИЕ

В таблице приведены сравнительные характеристики всех вышеописанных алгоритмов.

К недостаткам большинства из них стоит отнести недостаточно высокую вычислительную устойчивость, нерешенные численные проблемы, связанные с округлением. Многие алгоритмы реализуют набор операций, не являющийся замкнутым, так как в результате их выполнения могут получаться контуры с самокасаниями, что неприемлемо в качестве исходных данных.

Одним из наиболее простых алгоритмов можно признать алгоритм О'Рурка, однако он обрабатывает только выпуклые многоугольники. Можно выделить также алгоритмы Леонова и Холверда – наряду с устойчивой и быстрой работой, способностью обрабатывать общие многоугольники, в них решены проблемы появления «посторонних пересечений», возникающих при округлении точек пересечения.

Однако из-за высокой скорости работы и лучшей вычислительной устойчивости для практического применения стоит рекомендовать триангуляционный алгоритм в случае относительно большого числа узлов исходных многоугольников и линейно-узловой алгоритм для меньшего числа.

Сравнительные характеристики различных алгоритмов построения оверлеев: n_1 – количество вершин первого многоугольника A; n_2 – второго (B); n_0 – количество новых точек пересечения сторон двух многоугольников; z – общее число контуров многоугольников A и B $n = n_1 + n_2 + n_0$

Алгоритм	Поддерживаемые операции			Тип исходных многоугольников	Простота программной реализации	Устойчивость	Скорость	Трудоемкость	Общая оценка
	$\cap$	$\cup$	/						
Сазерленда – Ходжмана	+	–	–	Один выпуклый, другой общий	5	3	4	$O(n_1 n_2 \log n_2)$	3
О'Рурка	+	+	+	Выпуклые	5	4	5	$O(n)$	5
Вейлера – Азертона	+	+	+	Простые с ориентированными контурами	3	3	4	$O(n\log(n_1+n_2))$	3
Леонова	+	+	+	Общие	4	5	4	$O((n_1+n_2)\log(n_1+n_2)+ +n_0+z\log(n_1+n_2))$	5
Шутте	+	–	+	Простые	4	3	4	$O(n_1 n_2)$	3
Холверда	+	+	+	Общие	3	5	4	$O(n\log n)$	4
Маргалита – Кнотта	+	+	+	Вершинно-полные	4	4	4	$O(n)$	4
Линейно-узловой	+	+	+	Общие	4	5	4	$O(n^2)$	5
Триангуляционный	+	+	+	Общие	5	5	5	$O(n)$	5

ЛИТЕРАТУРА

1. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение / Пер. с англ. М.: Мир, 1989. 478 с.
2. *Foley D.J., van Dam A., Feiner S.K., Hughes J.F.* Computer graphics. Principles and practice. N.Y.: Addisson-Westey, 1991.
3. *Роджерс Д.* Алгоритмические основы машинной графики / Пер. с англ. М.: Мир, 1989. 512 с.
4. *Weiler K., Atherton P.* Hidden surface removing using polygon area sorting // Computer Graphics. 1977. V.11. P. 214–222.
5. *Weiler K.* Polygon comparison using graph representation // Computer Graphics. 1980. V.14. P. 10–18.
6. *Schutte K.* An edge labeling approach to concave polygon clipping // ACM Transactions on Graphics. 1995. P. 1–10.
7. *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.
8. *Sutherland I.E., Hodgman G.W.* Reentrant polygon clipping // CACM. 1983. V.26. P. 868–877.
9. *Clark J.H.* A VLSI geometry Processor for Graphics // IEEE Computer. 1980. V.12 (7). P 59–68.
10. *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.
11. *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.
12. *Holwerda K.* Complete Boolean Description (http:// www.xs4all.nl/~kholwerd/bool.html).
13. *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.
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.

Статья представлена факультетом информатики Томского государственного университета, поступила в научную редакцию 11 мая 2003 г.