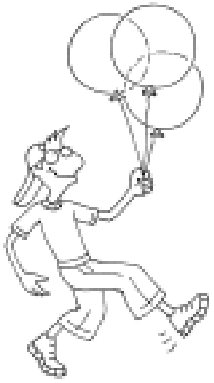
Ивановский Сергей Алексеевич,
Преображенский Алексей Семенович,
Симончик Сергей Константинович

АЛГОРИТМЫ ВЫЧИСЛИТЕЛЬНОЙ ГЕОМЕТРИИ. ВЫПУКЛЫЕ ОБОЛОЧКИ В ТРЕХМЕРНОМ ПРОСТРАНСТВЕ



1. ВВЕДЕНИЕ

Выпуклые оболочки в трехмерном пространстве используются в различных приложениях. Например, они используются для ускорения нахождения коллизий трехмерных объектов в компьютерной анимации. Предположим, требуется проверить, пересекаются ли два объекта P_1 и P_2 сложной формы. Если в большинстве случаев объекты не пересекаются, то следующая стратегия дает выигрыш. Сначала аппроксимируем объекты P_1 и P_2 более простыми объектами P_1^* и P_2^* соответственно, которые содержат в себе исходные объекты. Затем для проверки пересечения P_1 и P_2 сначала проверяем, пересекаются ли P_1^* и P_2^*. Только в случае, когда P_1^* и P_2^* пересекаются, проверяем, пересекаются ли исходные объекты.

Каким должен быть аппроксимирующий объект? С одной стороны, аппроксимирующие объекты должны быть простыми, чтобы проверки пересечения были дешевыми. С другой стороны, простая аппроксимация не дает возможности хорошо приблизить исходный объект, вследствие чего с большой вероятностью придется выполнять проверку пересечения исходных объектов.

В случае ограничивающих сфер проверка пересечения сфер довольно проста, но для многих объектов сферы не дают хорошей аппроксимации. При использовании выпуклых оболочек объектов в качестве их аппроксимации проверка пересечения более сложна, чем для сфер, но зато большинство объектов лучше аппроксимируются, и факт отсутствия пересечений близлежащих объектов устанавливается с меньшими затратами (см. рис. 1).

Алгоритмы построения выпуклых оболочек на плоскости были описаны в [1, 2]. Некоторые из них могут быть обобщены на трехмерный случай.

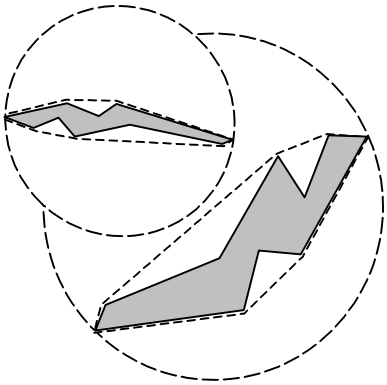


Рис. 1. Аппроксимация объектов ограничивающими сферами и выпуклыми оболочками

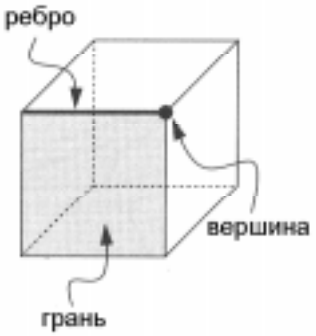


Рис. 2. Пример многогранника

2. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

По определению *многогранник* – это *тело*, ограниченное (со всех сторон) конечным числом плоскостей. Многоугольники, образованные пересечением этих плоскостей, называют *гранями*, их стороны – *ребрами*, а их вершины – *вершинами* многогранника (см. рис. 2). Поверхность многогранника, образованную гранями, также называют многогранником (обычно из контекста ясно, идет ли речь о поверхности или о теле).

Более точно определим многогранник [3] как множество (конечное) многоугольников, связанных таким образом, что, во-первых, при каждом ребре сходятся (под некоторым углом) два и только два многоугольника, а во-вторых, от всякого многоугольника можно прейти к другому, переходя через ребра.

В соответствии с данным определением, фигуры, представленные на рис. 3 *а–д*, являются многогранниками, а на рис. 3 *е–з* – нет.

Выпуклым называется многогранник, расположенный по одну сторону каждой из своих граней. Выпуклый многогранник можно положить на плоскость (например, на плоскость стола) на любую из его граней. Многогранники на рис. 3 *а–в* выпуклые, а на рис. 3 *г*, *д* – нет. Грань выпуклого многогранника представляет собой выпуклый многоугольник.

Как и в двумерном случае, *выпуклая оболочка* $CH(Q)$ множества точек Q в трехмерном пространстве – это наименьшее выпуклое множество, включающее в себя все эти точки. Выпуклая оболочка конечного множества точек в трехмерном пространстве является выпуклым многогранником, вершины которого принадлежат исходному множеству точек.

3. СЛОЖНОСТЬ ПРЕДСТАВЛЕНИЯ ВЫПУКЛОЙ ОБОЛОЧКИ

Многогранник может быть полностью определен перечислением его вершин, ребер и граней. Однако удобство и эффективность

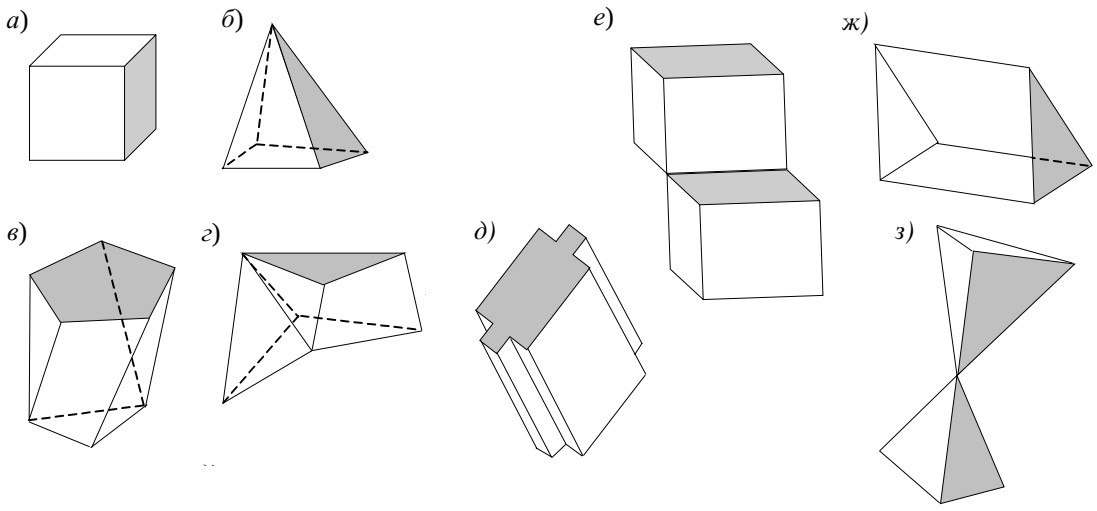


Рис. 3. Фигуры, ограниченные плоскостями

работы с таким объектом в алгоритмах зависит от способа представления его в виде некоторой структуры данных. Количественно сложность такого представления можно охарактеризовать объемом необходимой памяти, который будет зависеть от степени сложности многогранника, то есть от количества его вершин n, количества его ребер m и количества его граней l. Оказывается, что эти три параметра выпуклого многогранника в трехмерном пространстве связаны формулой Эйлера $n - m + l = 2$. Геометрическое доказательство этого факта можно найти, например, в [3], [4].

Удобно использовать преобразование выпуклого многогранника в связный плоский (планарный) граф (такой граф укладывается на плоскости без самопересечений, при этом все его ребра – прямолинейные отрезки [5]). Для этого выберем некоторую грань многогранника. Будем рассматривать ее как верхнюю. Возьмем внутреннюю точку этой грани (выпуклого многоугольника) и поднимемся из нее вверх по нормали к грани до некоторой точки p. Рассмотрим множество лучей, исходящих из точки p и проходящих через вершины многогранника (считаем, что точка p выбрана так, чтобы все эти лучи пересекали выбранную верхнюю грань в ее внутренних точках, кроме лучей проходящих через вершины этой грани). Рассмотрим точки пересечения этих лучей с плоскостью, параллельной выбранной грани и лежащей ниже многогранника или касающейся его. Полученные точки будем считать вершинами графа. Тогда его ребрами будут отрезки, являющиеся проекциями ребер многогранника (см. пример на рис. 4).

Полученный плоский прямолинейный граф разбивает плоскость на множество выпуклых многоугольников (граней разбиения плоскости). Поскольку формула Эйлера справедлива для произвольного связного плоского графа [5], то она с учетом описанного преобразования справедлива и для выпуклого многогранника. Более того, из формулы Эйлера (с учетом того, что каждая грань и вершина многогранника имеет, по крайней мере, три инцидентных ребра) следуют соотношения, связывающие попарно параметры выпуклого многогранника [5]:

$$m \le 3n - 6; \quad l \le 2n - 4;$$

$$n \le 3l - 4; \quad m \le 3l - 6.$$

Таким образом для полного описания (представления) многогранника с n вершинами достаточно объема памяти $O(n)$.

Преобразование выпуклого многогранника в плоский граф дает возможность использовать для представления выпуклого многогранника структуры данных, подходящие для представления планарного графа (такие, как списки смежности или реберный список с двойными связями [6]). Представление с помощью реберного списка рассмотрено в приложении. Важно от-

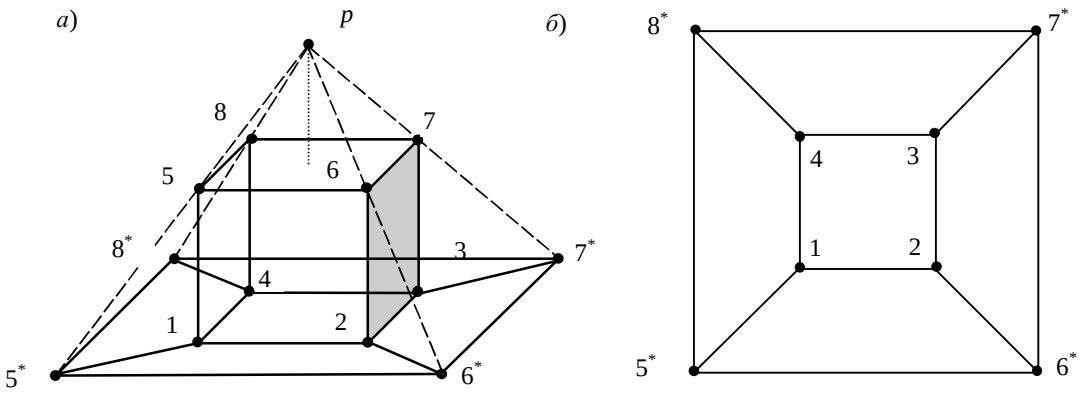


Рис. 4. Пример преобразования выпуклого многогранника (куба) в плоский граф:
а) проектирование куба: p – центр проекции; грань куба (1, 2, 3, 4) лежит в плоскости проекции;
б) плоский прямолинейный граф; верхняя грань куба (5, 6, 7, 8) преобразуется во внешнюю (бесконечную) область – вне квадрата (5^*, 6^*, 7^*, 8^*)

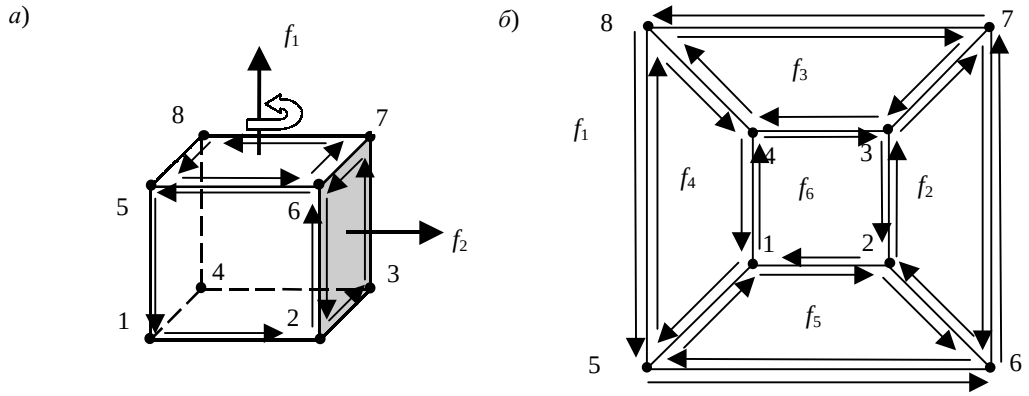


Рис. 5. Представление *а*) куба и
б) плоского графа ориентированными гранями и ориентированными ребрами

метить, что при таком представлении (и использовании его в алгоритмах) полезно ввести определенную ориентацию ребер и граней. Поясним это с помощью рис. 5.

Каждая грань куба (как отдельный многоугольник) ориентирована так, что нормаль к ней направлена вне куба. На рис. 5 *а* показаны нормали к граням f_1 и f_2. Если смотреть на грань со стороны нормали, то ребра ориентированы в порядке обхода против часовой стрелки. Представление граней изображенного на рис. 5 куба списками вершин и ориентированных ребер показано в табл. 1.

Каждое ребро куба представлено в гранях дважды. Причем его ориентация различна в соседних гранях (в списках ориентированных ребер). Например, неориентированное ребро куба $\{p_6, p_7\}$ представлено в гранях f_1 и f_2 ориентированными ребрами $[p_6, p_7]$ и $[p_7, p_6]$, соответственно. Общее количество всех ориентированных ребер равно удвоенному количеству ребер в выпуклом многограннике.

Отметим, что на рис. 5 *б* плоский граф изображен при взгляде на плоскость проекции сверху, поэтому его грани обходятся *по часовой* стрелке (за исключением внешней грани f_1).

Выпуклый многогранник называется *симплициальным*, если каждая его грань представляет собой треугольник. Работа с такими многогранниками в алгоритмах осуществляется более эффективно. Произвольный выпуклый многогранник можно представлять симплициальным, если произвести триангуляцию его граней (выпуклых многоугольников). В описываемых далее алгоритмах выпуклая оболочка будет строиться в виде симплициального многогранника.

После того как установлена сложность представления выпуклой оболочки, естественно задаться вопросом, какова нижняя оценка сложности построения выпуклой оболочки в трехмерном пространстве? Так как любая совокупность точек в двумерном

Табл. 1

Грань	Список вершин	Список ориентированных ребер
f_1	(p_5, p_6, p_7, p_8)	$([p_5, p_6], [p_6, p_7], [p_7, p_8], [p_8, p_5])$
f_2	(p_2, p_3, p_7, p_6)	$([p_2, p_3], [p_3, p_7], [p_7, p_6], [p_6, p_2])$
f_3	(p_3, p_4, p_8, p_7)	$([p_3, p_4], [p_4, p_8], [p_8, p_7], [p_7, p_3])$
f_4	(p_1, p_5, p_8, p_4)	$([p_1, p_5], [p_5, p_8], [p_8, p_4], [p_4, p_1])$
f_5	(p_1, p_2, p_6, p_5)	$([p_1, p_2], [p_2, p_6], [p_6, p_5], [p_5, p_1])$
f_6	(p_1, p_4, p_3, p_2)	$([p_1, p_4], [p_4, p_3], [p_3, p_2], [p_2, p_1])$

пространстве тривиальным образом вкладывается в трехмерное пространство, то нижняя оценка сложности построения выпуклой оболочки в трехмерном пространстве равна $O(n \log n)$. Ниже будут представлены алгоритмы, достигающие нижней оценки.

4. АЛГОРИТМ «ЗАВОРАЧИВАНИЯ ПОДАРКА»

Основная идея алгоритма [7] заключается в последовательном построении выпуклой оболочки путем переходов от одной грани (уже существующей) к смежной с ней новой грани, примерно так, как это происходит при заворачивании выпуклого ограниченного плоскими гранями объекта в лист бумаги (см. рис. 6). По сути это та же идея, что и в алгоритме Джарвиса, рассмотренном ранее [1], но приспособленная к трехмерному случаю.

Основой метода является то, что в выпуклой оболочке любое ребро является общим в точности для двух граней, и две грани имеют общее ребро тогда и только тогда, когда ребро определяется общим подмножеством из двух вершин для множеств вершин, определяемых этими гранями.

На каждом шаге строится связная часть выпуклой оболочки (из одной грани можно пройти в другую, если у них есть общее ребро). Выбирается грань f из текущей построенной части выпуклой оболочки, у этой грани выбирается ребро e такое, что другая смежная по этому ребру грань еще не найдена. Строится плоскость p, содержащая в себе грань f. Построенная плоскость «наклоняется» через ребро e к множеству точек так, чтобы выбрать «подходящую» точку p. На базе ребра e и точки p строится новая грань выпуклой оболочки, и если это не последняя грань, то алгоритм продолжает работу. Как и в двумерном случае, подходящую точку p можно охарактеризовать наименьшим углом поворота плоскости p (см. рис. 7).

Формализуем выходные данные алгоритма. Выпуклая оболочка будет представляться симплициальным многогранником. В данном алгоритме нет необходимости использовать в явном виде реберный список с двойными связями. Можно обойтись заданием связанных друг с другом множества вершин, множества ориентированных ребер и множества ориентированных треугольных граней. Как и в разделе 3, будем считать, что вершины ориентированной грани выпуклой оболочки перечислены против часовой стрелки, если смотреть на грань извне.

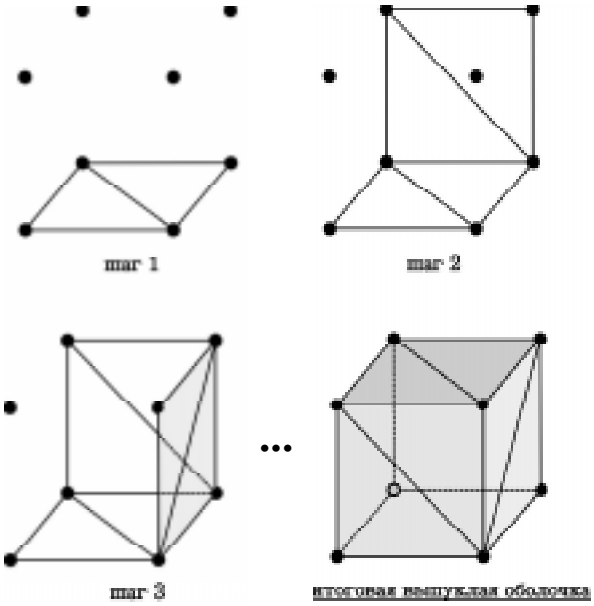


Рис. 6. Несколько шагов построения выпуклой оболочки методом заворачивания подарка

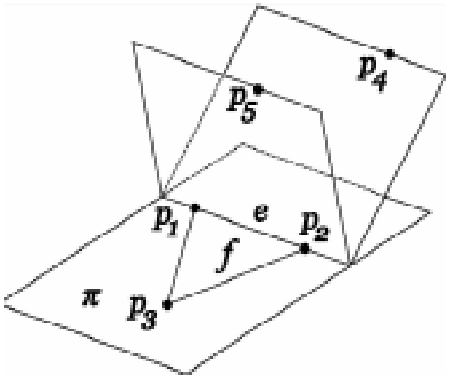


Рис. 7. Иллюстрация выбора грани, определяемой ребром e и точкой $p = p_4$ и образующей наименьший выпуклый угол с плоскостью, содержащей грань f

Далее при записи действий алгоритма нам понадобится понятие векторного произведения трехмерных векторов. Векторным произведением [8] вектора $\vec{a}$ и вектора $\vec{b}$ называется вектор $\vec{c}$, обозначаемый как $\vec{c} = [\vec{a}, \vec{b}]$ и удовлетворяющий трем условиям:

1. Длина вектора $|\vec{c}| = |\vec{a}||\vec{b}| \sin\varphi$ (есть площадь параллелограмма, см. рис. 8);

2. Вектор $\vec{c}$ ортогонален к каждому из векторов $\vec{a}$ и $\vec{b}$;

3. Вектор $\vec{c}$ направлен так, что тройка векторов $\vec{a}$, $\vec{b}$ и $\vec{c}$ является *правой* (см. рис. 8).

В декартовых координатах $\vec{c} = [\vec{a}, \vec{b}] = (y_a z_b - y_b z_a,\ z_a x_b - z_b x_a,\ x_a y_b - x_b y_a)$

С помощью векторного произведения легко связать нормаль ориентированной грани и ее ориентированные ребра. Например, на рис. 9 нормаль грани f_1 тетраэдра равна $[\overrightarrow{ab}, \overrightarrow{bd}]$ и направлена на нас, а нормаль грани f_2 равна $[\overrightarrow{bc}, \overrightarrow{cd}]$.

Определим для плоскости π *верхнее полупространство* как $\{p \in R^3 | (p - p_0, n) > 0\}$, где $p_0 \in \pi$ и n – нормаль π и (a,b) обозначает скалярное произведение векторов $\vec{a}$ и $\vec{b}$. *Нижнее полупространство* определим как R^3 \ верхнее полупространство. Рис. 10 иллюстрирует эти определения.

После этих предварительных определений рассмотрим набросок алгоритма (листинг 1).

Поясним и проанализируем основные шаги алгоритма.

Построение начального ребра (строка 2), принадлежащего выпуклой оболочке $CH(P)$ может быть сделано за время $O(n)$ следующим образом.

1. Найти точку $p_0 \in P$, заведомо являющуюся вершиной выпуклой оболочки, в качестве такой точки можно взять наименьшую точку, если на точках определен следующий порядок: для точек $p_1, p_2 \in R^3$ $p_1 < p_2$, если вектор p_1 лексикографически меньше вектора p_2 (сложность шага $O(n)$).

2. Построить плоскость π, $p_0 \in \pi$, $\pi \| YOZ$ (сложность шага $O(n)$).

3. Плоскость π наклоняется через прямую, параллельную оси OY и проходящую через точку p_0, до тех пор, пока не будет встречена первая точка p_1 (сложность шага $O(n)$).

4. Сформировать множество точек Q, принадлежащих плоскости π ($p_0, p_1 \in Q$) за время $O(n)$.

5. Найти любое ребро выпуклой оболочки множества точек Q (аналогично тому, как это сделано в алгоритме Джарвиса) за время $O(n)$. Это ребро будет ребром $CH(P)$.

В процессе работы алгоритма используется стек ориентированных ребер. Таким образом, если пара вершин a и b определяет ребро $CH(P)$, то в стек на разных шагах алгоритма будут добавлены ориентированные ребра $[a,b]$ и $[b,a]$. Ориентированность ребер позволяет однозначно определить грань выпуклой оболочки и сориентировать её правильным образом.

В 5-ой строке алгоритма нормаль плоскости π определяется как $[\overrightarrow{ab}, \overrightarrow{ap_i}]$ (ребро (a,b) ориентировано в порядке обхода вер-

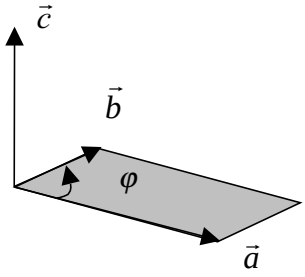


Рис. 8. Векторное произведение $\vec{c} = [\vec{a}, \vec{b}]$ вектора $\vec{a}$ и вектора $\vec{b}$. Со стороны вектора $\vec{c}$ угол φ ($0 \le \varphi \le \pi$) отсчитывается против часовой стрелки

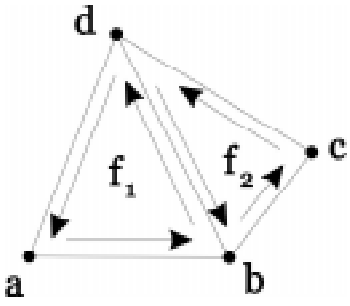


Рис. 9. Ориентированные ребра, инцидентные грани f_1, есть $[a,b]$, $[b,d]$ и $[d,a]$, а нормаль f_1 равна $[\overrightarrow{ab}, \overrightarrow{bd}]$; грани f_2 инцидентны ориентированные ребра $[b,c]$, $[c,d]$ и $[d,b]$, а нормаль f_2 равна $[\overrightarrow{bc}, \overrightarrow{cd}]$

шин содержащейся в плоскости π грани против часовой стрелки, если смотреть извне на выпуклую оболочку). Таким образом, шаги 5 и 6 могут быть сделаны за время $O(n)$.

На шаге 7 строится двумерная выпуклая оболочка найденного множества точек T (в плоскости π могут лежать более трех точек). Этот шаг можно сделать за время $O\left(|T|\cdot|T_{extreme}|\right)$, где $T_{extreme}$ – множество крайних точек выпуклой оболочки множества T. Для достижения такой оценки можно использовать алгоритм Джарвиса, либо асимптотически оптимальный алгоритм со сложностью $O\left(|T|\log\left(|T_{extreme}|\right)\right)$. При анализе общей сложности алгоритма важно, чтобы на этом шаге использовался алгоритм, сложность которого определяется числом вершин построенной оболочки $T_{extreme}$.

После построения двумерной выпуклой оболочки $CH(T)$ определен обход её вершин (против часовой стрелки, если смотреть извне). Следовательно, можно выделить и сориентировать компланарные треугольные грани (строка 8), принадлежащие π (см. рис. 11), за линейное по количеству этих граней время (таких граней будет $|CH(T)|-2$). Также за линейное время можно выделить и добавить в стек ориентированных ребер крайние ребра $CH(T)$ для дальнейшего построения оболочки (исключая ребро (a,b) их будет $|CH(T)|-1$).

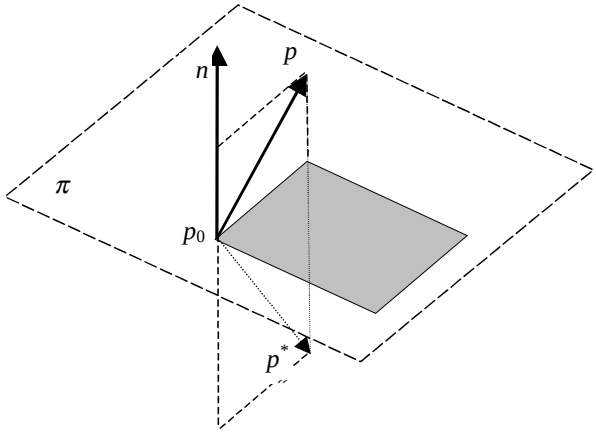


Рис. 10. Точка p лежит в верхнем полупространстве плоскости π (здесь $(p-p_0,n)>0$), а точка p^* – в нижнем полупространстве (здесь $(p*-p_0,n)<0$); для всех точек q плоскости π справедливо $(p-p_0,n)=0$

Узнать (строка 9), было ли ориентированное ребро ранее добавлено в стек, мож-

Листинг 1. Algorithm GIFT-WRAPPING(P) $\rightarrow$ CH(P)

Вход. Исходное множество точек $P=\left\{p_i \,|\, p_i=(x_i,y_i),\ i=[1..n]\right\}$.

Выход. Выпуклая оболочка множества точек $CH(P)$, заданная набором треугольных граней.

1 инициализировать выпуклую оболочку пустой
2 найти некоторое исходное ребро (с произвольной ориентацией) результирующей выпуклой оболочки и добавить его в стек ориентированных ребер S
3 **while** Size(S) > 0 **do**
4 $e \leftarrow$ Pop(S)
5 найти такую плоскость π, содержащую $CH(e\cup p)$, что $p\in P$ и все точки $p_i\in P$ принадлежат нижнему полупространству плоскости π; при этом если $e=[a,b]$, то нормаль π определяется как $[\overrightarrow{ab},\overrightarrow{ap_i}]$
6 найти множество $T=\left\{p\in P \,|\, p\in\pi\right\}$
7 построить двумерную выпуклую оболочку $CH(T)$
8 создать новые треугольные грани, определяемые $CH(T)$, и добавить их в $CH(P)$
9 обойти новые крайние ребра $CH(P)$ (ребра $CH(T)$), и добавить их в стек S, если они еще не были добавлены ранее
10 **end-do**
11 вернуть построенную выпуклую оболочку $CH(P)$

algorithm UPPERHULL (Q) $\rightarrow$ верхняя оболочка

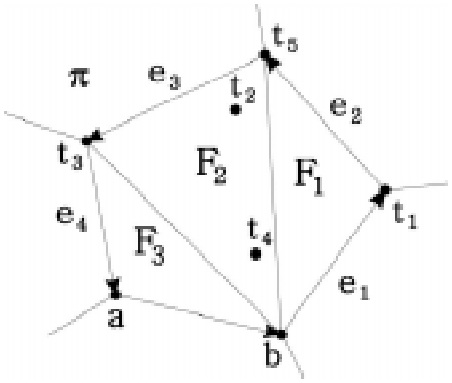


Рис. 11. В плоскости π лежат точки $T=\{a,b,t_1,t_2,t_3,t_4,t_5\}$. Их выпуклая оболочка есть $CH(T)=\{a,b,t_1,t_5,t_3\}$. Созданы треугольные грани $F_1=(b,t_1,t_5)$, $F_2=(b,t_5,t_3)$, $F_3=(b,t_3,a)$, $F_1,F_2,F_3\in\pi$. В стек добавлены ориентированные ребра $e_1=(b,t_1)$, $e_2=(t_1,t_5)$, $e_2=(t_5,t_3)$, $e_4=(t_3,a)$

но за время $O(\log(n))$, используя сбалансированное дерево поиска [9] и введя произвольный порядок на ребрах. Заметим, что поскольку необходимо только добавлять ребра в структуру данных и искать в ней, то в качестве такой структуры можно использовать хеш-таблицу [9].

Общее количество выполнений цикла **while** (строки 3–10) равно количеству граней F, принадлежащих разным плоскостям многогранника.

Оценим общую сложность алгоритма. Обозначим за T^i множество T, а за $T^i_{extreme}$ множество $T_{extreme}$ на i-ом шаге работы алгоритма (другими словами при нахождении i-ой плоскости). Поиск множества T^i можно выполнить за $O(n)$ операций (строки 5–6). Построение двумерной выпуклой оболочки делается за $O\left(|T^i|\cdot|T^i_{extreme}|\right)$ операций (строка 7). Создание новых $O\left(|T^i_{extreme}|\right)$ граней требует линейного времени (строка 8), поиск и добавление $O\left(|T_{extreme}|\right)$ ориентированных ребер в стек требует $O\left(|T_{extreme}|\cdot\log(F)\right)$.

Пусть L – количество ребер в $CH(P)$, тогда

$$\sum_{i=1}^{F}|T^i_{extreme}|=2\cdot L\in O(F)\ \text{и}$$

$$\sum_{i=1}^{F}|T^i|\le n+\sum_{i=1}^{F}|T^i_{extreme}|\in O(n).$$

Общее время работы:

$$\sum_{i=1}^{F}\left(O(n)+O\left(|T^i|\cdot|T^i_{extreme}|\right)+\right.$$
$$\left.+O\left(|T^i_{extreme}|\right)+O\left(|T^i_{extreme}|\cdot\log(F)\right)\right)\le$$
$$\le O(nF)+\sum_{i=1}^{F}O\left(|T^i|\cdot|T^i_{extreme}|\right)+O(F)+$$
$$+O\left(F\log(F)\right)\le O(nF)+O\left(nF\right)\le O(nF)$$

Итак, в среднем время работы алгоритма составляет $O(nF)$, а в худшем случае, когда все точки множества оказываются на оболочке, – $O(n^2)$.

5. АЛГОРИТМ РАЗДЕЛЕНИЯ И СЛИЯНИЯ

Нижняя оценка для задачи построения выпуклой оболочки в трехмерном пространстве такая же, как и в двумерном случае: $\Omega(n\log n)$. Алгоритм сбалансированного разделения и слияния, основанный на методе «разделяй и властвуй» (*лат.* «Divide et impera», *англ.* «divide-and-conquer»), был предложен в [10] и достигает нижней оценки. Позже в алгоритм были внесены важные корректировки [6], а в [11] описана реализация алгоритма. Алгоритм является обобщением алгоритма сбалансированного разделения и слияния «с мостиками» для плоского случая [2].

Принцип алгоритма такой же, как и в двумерном пространстве: отсортировать множество точек по x координате, разбить его на два линейно разделимых множества, рекурсивно построить выпуклые оболочки в каждом из множеств и затем слить их в одну оболочку. Слияние может быть сделано за $O(n)$ операций, и, таким образом, общая сложность составляет $O(n\log n)$.

В листинге 2 приводится укрупненный псевдокод алгоритма с учетом того, что на

вход подается отсортированное по координате x множество точек.

Предварительная сортировка элементов множества S по координате x требует $O(n \log n)$ операций. Благодаря сортировке и разделению на шагах 5–6 алгоритма, множества $CH(P_1)$ и $CH(P_2)$ представляют два непересекающихся трехмерных выпуклых многогранника. Основная сложность заключается в выполнении функции $Merge(A, B)$ за время $O\left(|A|+|B|\right)$. На этом шаге необходимо построить цилиндрическую триангуляцию (см. рис. 12), опирающуюся на выпуклые оболочки A и B по некоторым контурам, и удалить из A и B части, оказавшиеся скрытыми. К выпуклой оболочке $CH(A \cup B)$ добавится некоторый «обод» из граней с топологией цилиндра без оснований. Количество граней линейно зависит от размера двух многогранников: каждая грань использует как минимум одно ребро из A или из B. Таким образом, количество граней не превосходит общего количества ребер.

Рассмотрим шаг слияния двух оболочек более подробно: сначала будет описана процедура слияния двух выпуклых оболочек из [6], а затем будут предложены улучшения.

Рассмотрим слияние выпуклых оболочек.

В [6] для представления структуры выпуклой оболочки используется реберный список с двойными связями (РСДС). Укрупнено шаг слияния можно представить следующим образом:

1. Построить цилиндрическую триангуляцию Г, опирающуюся на A и B.

2. Удалить из A и B части, оказавшиеся скрытыми в результате построения триангуляции Г.

Основываясь на интуитивных соображениях, построение триангуляции Г можно рассматривать как операцию заворачивания одновременно двух «подарков» в один сверток. Хотя Г может иметь $O(n)$ граней, а каждый шаг заворачивания в общем случае требует $O(n)$ операций. Использование особенностей трехмерных многогранников позволяет решить эту задачу за линейное время.

Построение триангуляции начинается с нахождения некоторого ее ребра. Наиболее удобный способ состоит в проектировании выпуклых оболочек A и B на плоскость XOY и затем нахождении нижнего мостика e (таким образом можно поддерживать нижнюю выпуклую оболочку проекции точек на XOY) (см. рис. 13). Имея ребро e, можно начать построение Г, выбрав в качестве опорной плоскость, проходящую через ребро e параллельно оси OZ.

На очередном шаге построения Г в качестве базы используется последняя из уже

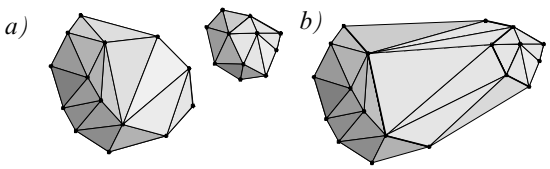


Рис. 12. Слияние непересекающихся выпуклых оболочек:
а) выпуклые многогранники до соединения;
б) результирующая выпуклая оболочка.
Жирные ребра показывают границу новых граней

Листинг 2. Algorithm DIVIDE-AND-CONQUER(P) $\rightarrow CH$(P)

```
1 | if (|P| ≤ k0) then
2 |    построить CH(P) любым методом
3 |    вернуть построенную CH(P)
4 |        end-if
5 |  P1 ← {p1,..., p⌊n/2⌋};
6 |  P2 ← {p⌊n/2⌋+1,..., pn}
7 |  A ← CH(P1); B ← CH(P2);
8 |  H ← Merge(A, B);
9 | вернуть H в качестве построенной CH(P)
```

построенных граней триангуляции Г. Пусть грань (a_2, b_2, a_1) является базовой на текущем шаге. Далее среди вершин, смежных с a_2 необходимо выбрать вершину $\hat{a}$ так, чтобы грань $(a_2, b_2, \hat{a})$ образовала наибольший выпуклый угол с (a_2, b_2, a_1) среди всех граней (a_2, b_2, v) для вершин v, смежных с a_2 и $v \neq a_1$.

Аналогичным образом среди всех вершин, смежных с b_2 выберем вершину $\hat{b}$. Теперь, когда выявлены два претендента $(a_2, b_2, \hat{a})$ и $(a_2, b_2, \hat{b})$, делается заключительное сравнение: если $(a_2, b_2, \hat{a})$ образует с (a_2, b_2, a_1) больший выпуклый угол, чем $(a_2, b_2, \hat{b})$, то $\hat{a}$ добавляется к Г (в противном случае добавляется $\hat{b}$), и на этом шаг построения очередной треугольной грани заканчивается. Используя тот факт, что в РСДС можно эффективно проходить ребра, инцидентные некоторой вершине в порядке обхода по часовой стрелке или против нее, можно представлять информацию о пройденных вершинах для выпуклых оболочек следующим образом (см. рис. 14).

Пусть грань (b_s, b, a) образует наибольший выпуклый угол с (b_1, b, a) из всех (b_i, b, a) с $i = 2, \ldots, k$. Любое ребро (b_i, b) при $1 < i < s$ оказывается внутри выпуклой оболочки $CH(P_1 \cup P_2)$, и поэтому его можно исключить из дальнейшего рассмотрения. Просмотр ребер, инцидентных a, следует начинать с последнего из просмотренных ребер, так как все другие удалены. Таким образом, на заворачивание требуется $O(\text{ребер в } CH(P_1 \cup P_2))$ операций.

Удаление из A и B частей, оказавшихся скрытыми, можно сделать за линейное время от размера удаляемых частей (процесс удаления можно охарактеризовать как обход структуры, начиная с заведомо удаляемой грани, и не переходящий через ребра, помеченные как принадлежащие «ободам» цилиндрической триангуляции).

Таким образом, слияние $CH(P_1)$ и $CH(P_2)$ может быть выполнено за время $O(|CH(P_1 \cup P_2)|)$.

Как показывает опыт реализации алгоритма, построение цилиндрической триангуляции можно упростить, если хранить неполную информацию из РСДС: можно исключить информацию о ребрах выпуклой оболочки. Для построения цилиндрической триангуляции достаточно информации о структуре многогранников, состоящей из ссылок между гранями (каждая грань имеет три ссылки на грани смежные по ребрам, перечисленные в порядке обхода по часовой стрелке).

Таким образом, вместо того, чтобы обходить ребра, инцидентные вершине в порядке обхода против часовой стрелки, достаточно обойти грани сливаемых выпуклых оболочек, имеющие общие ребра с ободом цилиндрической триангуляции, и которые будут скрыты после слияния (то есть внут-

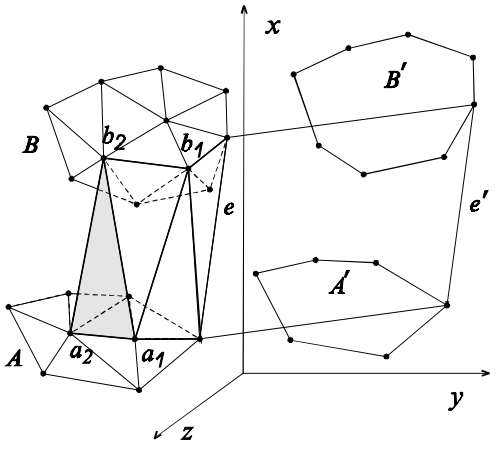


Рис. 13. Начальный и последующие шаги процедуры построения

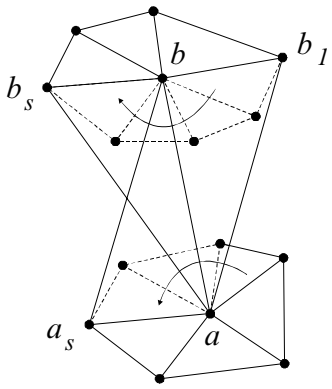


Рис. 14. Шаг, обеспечивающий продвижение при построении триангуляции Г ($s = 4$).

ренние грани). Обход граней производится по ободу цилиндрической триангуляции (см. рис. 15).

Пройденных граней будет не более чем общее количество граней в сливаемых оболочках, и, следовательно, общая сложность слияния есть $O(n)$.

Заметим, что триангуляция Г не может быть однозначно идентифицирована в тех случаях, когда грани цилиндрической триангуляции компланарны с гранями сливаемых выпуклых оболочек (например, при распределении точек на кубе этот случай встречается довольно часто). Это может создать проблемы с построением Г [11] (цилиндрическая триангуляция не всегда строится односвязной). На рис. 16 проиллюстрирована эта ситуация.

Реализация [11] использует внешний обод, однако удобнее строить цилиндрическую триангуляцию для внутреннего обода и, таким образом, упростить обработку частных случаев. Для выявления кандидата на включение в триангуляцию на следующем шаге, в случае компланарности граней-кандидатов, нужно брать ту грань, которая полностью принадлежит другой грани, если же ни одна не входит в другую, то можно брать произвольную.

После построения цилиндрической триангуляции можно обновить структуру РСДС за линейное время. Заметим, что при таком подходе к построению цилиндрической триангуляции существенно упрощается удаление скрытых частей сливаемых оболочек.

Размер необходимой алгоритму памяти определяется тем, что при хранении сливаемых выпуклых оболочек и цилиндрической триангуляции требуется хранить примерно $4n$ граней в худшем случае, что превосходит в два раза затраты на хранение просто граней выпуклой оболочки.

6. БЫСТРЫЙ АЛГОРИТМ

На содержательном уровне этот метод последовательно обрабатывает по одной точке, назовем ее $p \in P$, и если p – внешняя точка по отношению к текущей выпуклой оболочке $CH(P)$, то из точки p строится опорный конус к текущей $CH(P)$ и

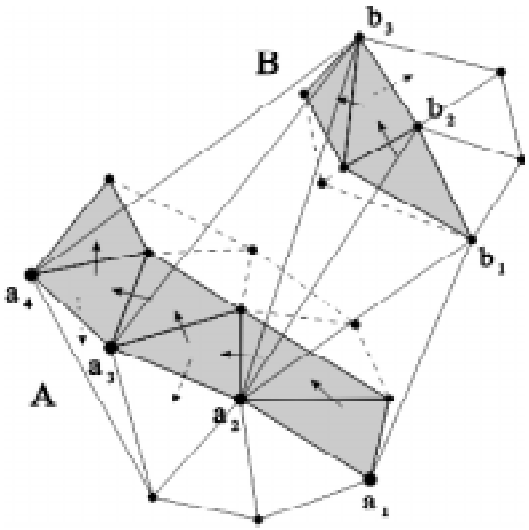


Рис. 15. Построение цилиндрической триангуляции при обходе граней, смежных с ободом цилиндрической триангуляции

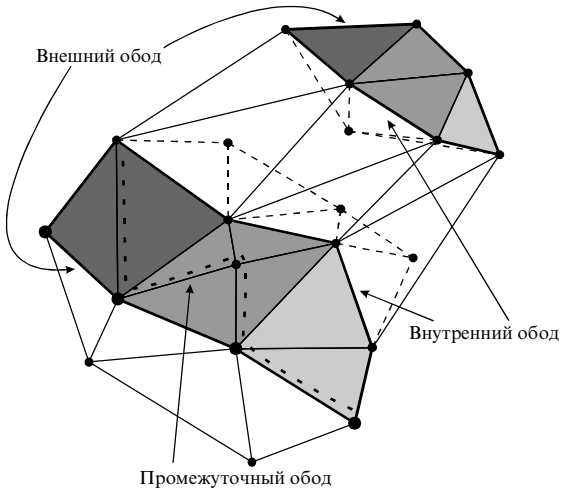


Рис. 16. Неоднозначность построения цилиндрической триангуляции Г. Грани, помеченные одним цветом, лежат в одной плоскости. Триангуляция Г может опираться как на внешний или внутренний обод, так и на любой из «промежуточных» ободов.

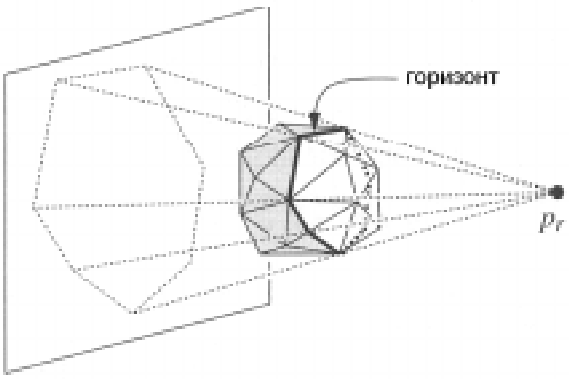


Рис. 17. Добавление точки p_r. Горизонт многогранника

удаляется часть оболочки $CH(P)$, затеняемая этим конусом.

Обозначим за P_r множество $\{p_1,..., p_r\}$, где $r \geq 1$. Рассмотрим шаг алгоритма добавления точки p_r к уже построенной выпуклой оболочке P_{r-1}. Иными словами, рассмотрим переход от $CH(P_{r-1})$ к $CH(P_r)$. Возможны два случая:

– точка p_r лежит внутри $CH(P_{r-1})$, либо на её границе, тогда $CH(P_r) = CH(P_{r-1})$.

– точка p_r лежит вне $CH(P_{r-1})$. Предположим, что в точке p_r находится точечный источник света (см. рис. 17). Некоторые грани $CH(P_{r-1})$ будут «освещены», а остальные грани будут «в тени». Освещенные или *видимые* грани формируют связанную область на поверхности выпуклой оболочки $CH(P_{r-1})$, называемую *видимым регионом* точки p_r на $CH(P_{r-1})$. Видимый регион ограничен замкнутой кривой, состоящей из ребер $CH(P_{r-1})$, в дальнейшем называемой *горизонтом* точки p_r на $CH(P_{r-1})$. Проекция горизонта является границей выпуклого многоугольника, полученного проецированием $CH(P_{r-1})$ на плоскость с центром проекции в точке p_r.

Листинг 3. Algorithm QUICKHULL(P) → *CH*(P)

Вход. Исходное множество точек

Выход. Выпуклая оболочка множества точек *CH(P)*, заданная набором треугольных граней.

```
1  | найти четыре точки p1, p2, p3, p4 из множества P, образующие
       тетраэдр с ненулевым объемом
2  | C ← CH({p1, p2, p3, p4})
3  | for каждой грани F из CH({p1, p2, p3, p4})
4  |     for каждой не отнесенной точки p
5  |       if p находится над F then
6  |         отнести p ко внешнему множеству F
7  | for каждой грани F из C с непустым внешним множеством
8  |     выбрать самую дальнюю точку p из внешнего множества F
9  |     инициализировать видимое множество V в F
10 |     for каждой непомеченной грани N, являющийся соседом грани из F
11 |       if p находится над N then
12 |         добавить N в V
13 |     преобразовать границу V в горизонт H
14 |     for каждого ребра R из H
15 |       создать новую грань из R и p
16 |       проставить ссылки новой грани на её соседей
17 |     for каждой новой грани F'
18 |       for каждой не отнесенной точки q из внешнего множества грани из V
19 |         if q находится над F' then
20 |           отнести q ко внешнему множеству F'
21 |     удалить грани из V
22 | вернуть C
```

Горизонт точки p_r играет важную роль в преобразовании $CH(\mathrm{P}_{r-1})$ в $CH(\mathrm{P}_r)$: он представляет собой границу между частью поверхности, которая должна быть сохранена (невидимые грани), и частью поверхности, которая должна быть удалена (видимые грани).

Видимые грани должны быть замещены гранями, образованными точкой p_r и её горизонтом.

Центральной задачей алгоритма является эффективное определение опорного конуса для добавляемой точки *p*. Заметим, что поскольку каждая грань имеет ссылки на ее соседей, нахождение первой видимой грани из точки p_r позволяет быстро найти оставшиеся видимые грани (за линейное время от количества этих граней).

В быстром алгоритме для нахождения первой видимой грани используются *внешние множества* точек для каждой грани. Точка находится во внешнем множестве грани, если она находится над гранью, каждая точка находится только в одном внешнем множестве. Отличительной особенностью алгоритма (см. листинг 3) является добавление самой дальней точки из внешнего множества грани.

В [12] приводится эмпирический анализ алгоритма. Определяются условия балансировки, при которых гарантируется *хорошее* поведение алгоритма в случаях, когда все точки исходного множества входят в выпуклую оболочку. При выполнении условий балансировки алгоритм QUICKHULL в среднем работает за время $O(n \log n)$.

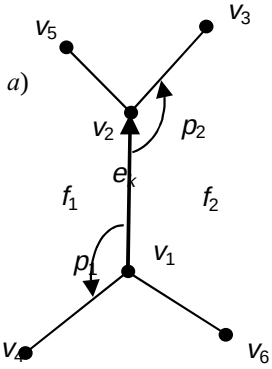
Приложение

РЕБЕРНЫЙ СПИСОК С ДВОЙНЫМИ СВЯЗЯМИ

Имея в виду эффективность реализации действий, типовых для многих алгоритмов, удобно представлять плоский прямолинейный граф с помощью реберного списка с двойными связями (РСДС) [6].

Пусть задан плоский граф $G = (V, E)$, где $V = \{v_1,...,v_N\}$ – вершины и $E = \{e_1,...,e_M\}$ – ребра. Главный элемент РСДС для плоского графа G – это *реберный узел*. Между ребрами графа и реберными узлами РСДС существует взаимно однозначное соответствие, то есть каждое ребро представлено в РСДС ровно один раз. Реберный узел РСДС (см. рис. 18), соответствующий ребру графа, например, $e_k = \{v_1, v_2\}$, имеет четыре информационных поля $(V1, V2, F1, F2)$ и два поля указателей $(P1, P2)$. Значения этих полей таковы. Поле $V1$ содержит *начало* ребра, а поле $V2$ содержит его *конец* (так изначально неориентированное ребро получает условную ориентацию, которая не несет пока никакой смысловой нагрузки). Поля $F1$ и $F2$ содержат имена граней, лежащих слева и справа от ориентированного ребра (v_1, v_2). Указатель $P1$ (соответственно $P2$) задает реберный узел, содержащий первое ребро, встречаемое вслед за ребром (v_1, v_2), при повороте от него против часовой стрелки вокруг v_1 (соответственно v_2).

Например, для графа, изображенного на рис. 19, реберный список и массивы *входов* в него по вершинам и граням приведены на рис. 20.



б)

ребро	$V1$	$V2$	$F1$	$F2$	$P1$	$P2$
e_k	v_1	v_2	f_1	f_2	p_1	p_2

Рис. 18. Представление плоского графа с помощью РСДС

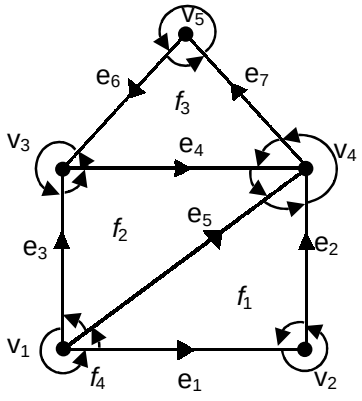


Рис. 19. Плоский граф, ребрам которого придана произвольная ориентация

а)	*б*)	*в*)
ребра	*V*	*F*
*V*1	*head_V*	*head_F*
*V*2		
*F*1	*v1*	*f1*
*F*2	1	1
*P*1		
*P*2	*v2*	*f2*
	2	2
e_1		
1	*v3*	*f3*
2	4	4
1		
4	*v4*	*f4*
5	7	7
2		
	v5	
e_2	6	
2		
4		
1		
4		
1		
7		
e_3		
1		
3		
4		
2		
1		
4		
e_4		
3		
4		
3		
2		
6		
5		
e_5		
1		
4		
2		
1		
3		
2		
e_6		

С помощью РСДС можно легко вычислить ребра, инцидентные заданной вершине следующим образом (см. листинг 4).

Время работы этой процедуры пропорционально числу ребер, инцидентных вершине *v*.

Обход заданной грани выполняется с помощью аналогичной процедуры (см. листинг 5).

В некоторых случаях полезно использовать более универсальный вариант РСДС, но требующий несколько большей памяти. Назовем его РСДС *с удвоением ребер*. В этом случае каждое неориентированное ребро исходного графа представляется парой ориентированных ребер (*полу*ребер). Такое представление поясняется на рис. 21. Именно оно, как правило, используется в описанных алгоритмах построения трехмерных выпуклых оболочек.

Листинг 4

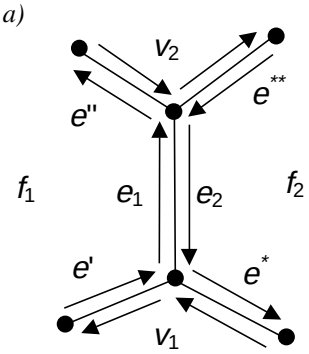
```
Proc Инцидентные_ребра (v: Вершина; var A: Вектор_ребер);
{A[1..*] – список ребер, инцидентных v, в порядке против часовой стрелки}
{a – текущее ребро, Список – РСДС }
var a, a0: ребро;  i: индекс;
begin
    a := Head_V[v];  a0 := a;  i := 0;
    {inv: ((Список[a].v1 = v) or (Список [a].v2 = v)) & A[1..i] – заполнен}
    repeat
        i := i + 1;
        A[i] := a;
        if  Список[a].v1 = v  then a := Список[a].p1
                              else  a := Список[a].p2;
    until a = a0;
end { Инцидентные_ребра }
```

Листинг 5

```
Proc Граница_грани (f: Грань; var A: Вектор_ребер);
{A[1..*] – список ребер границы грани в порядке по часовой стрелке}
{a – текущее ребро, Список – РСДС }
var a, a0: ребро;  i: индекс;
begin
    a := Head_F[v];  a0 := a;  i := 0;
    {inv: ((Список[a].f1 = f) or (Список [a].f2 = f)) & A[1..i] – заполнен}
    repeat
        i := i + 1;
        A[i] := a;
        if  Список[a].f1 = f  then a := Список[a].p1
                              else  a := Список[a].p2;
    until a = a0;
end { Граница_грани }
```

Рис. 20.
(*а*) РСДС,
(*б*) входы по вершинам *head_V* [1..*n*] и
(*в*) входы по граням *head_F*[1..*l*] для графа на рис. 19.

а)



б)

Полуребро	Вершина *V*	Парное ребро *E*	Грань слева *F*	Следующее ребро *P*1	Предыдущее ребро *P*2
e_1	v_1	e_2	f_1	e''	e'
e_2	v_2	e_1	f_2	e^*	e^{**}

Рис. 21. Представление плоского графа (*а*) с помощью РСДС с удвоением ребер (*б*).

Литература

1. *Ивановский С.А., Преображенский А.С., Симончик С.К.* Алгоритмы вычислительной геометрии. Выпуклые оболочки: простые алгоритмы // Компьютерные инструменты в образовании, 2007, №1. С. 4–19.

2. *Ивановский С.А., Преображенский А.С., Симончик С.К.* Алгоритмы вычислительной геометрии. Выпуклые оболочки: связь с задачей сортировки и оптимальные алгоритмы // Компьютерные инструменты в образовании, 2007, №2. С. 6–18.

3. *Гильберт Д., Кон-Фоссен С.* Наглядная геометрия, 1981. 344 с.

4. *Берже М.* Геометрия: в 2 т. Т. 1. М.: Мир, 1984. 560 с.

5. Лекции по теории графов /*Емеличев В.А. и др.* М.: Наука. Гл.ред.физ.-мат.лит., 1990. 384 с.

6. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение. М.: Мир, 1989. 478 с.

7. *D.R. Chand and S.S. Kapur.* An algorithm for convex polytopes, J. ACM, 17:78–86, 1970.

8. *В.А. Ильин, Э.Г. Позняк.* Аналитическая геометрия. М.: ФИЗМАТЛИТ, 2002. 240 с.

9. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦМНО, 2000. 960 с.

10. *F.P. Preparata and S.J. Hong.* Convex hulls of finite sets of points in two and three dimensions. Commun. ACM, 20:87–93, 1977.

11. *A.M. Day.* The implementation of an algorithm to find the convex hull of a set of three-dimensional points. ACM Trans. on Graphics, 9:105–132, 1990.

12. *B. Barber, D. Dobkin, and H. Huhdanpaa.* The quickhull algorithmfor convex hull. Technical Report GCG53, The Geometry Center, MN, 1993.

Ивановский Сергей Алексеевич, кандидат технических наук, доцент кафедры Математического обеспечения и применения ЭВМ СПбГЭТУ «ЛЭТИ»,

Преображенский Алексей Семенович, аспирант СПбГЭТУ «ЛЭТИ», магистр прикладной математики и информатики,

Симончик Сергей Константинович, аспирант СПбГЭТУ «ЛЭТИ» магистр прикладной математики и информатики.