

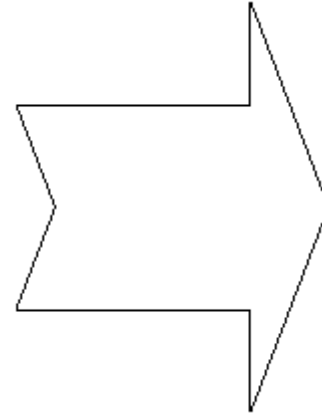
# Векторные алгоритмы

# План

- **Виды полигонов**
- Выпуклые оболочки
- Триангуляция
- Пересечение/объединение

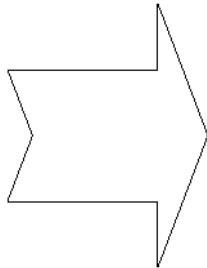
# Термины

- Полигон
- Рёбра и вершины
- Полигон называется *простым*, если он не пересекает самого себя.
- Внутренняя часть полигона
- Внешняя часть полигона
- Граница полигона
- Будем использовать термин *полигон* в смысле *простого заполненного полигона*, т.е. объединения границы и внутренней части простого полигона.



# Вершины

- Вершины упорядочены циклически вдоль границы полигона.
- Две вершины, являющиеся концевыми точками одного и того же ребра, являются *соседями* или *смежными* вершинами.
- Вершина, соседняя при обходе по часовой стрелке, называется *последователем*, а соседняя при обходе против часовой стрелки – ее *предшественником*.
- Вершина называется *выпуклой*, если внутренний угол при вершине (находящийся в пределах внутренней области)  $m$  или равен  $180$  градусов.
- В противном случае вершина считается *вогнутой* (внутренний угол больше  $180$  градусов).

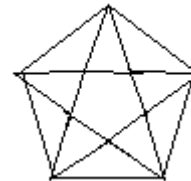
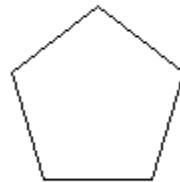
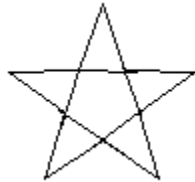
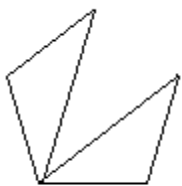


# Вырожденные полигоны

- Точку или отрезок прямой линии иногда удобно рассматривать как *вырожденный* полигон.
- 1-гон состоит из единственной вершины и имеет единственное ребро нулевой длины, соединяющей вершину саму с собой.
- 2-гон содержит две вершины и два совпадающих ребра, соединяющих две вершины.

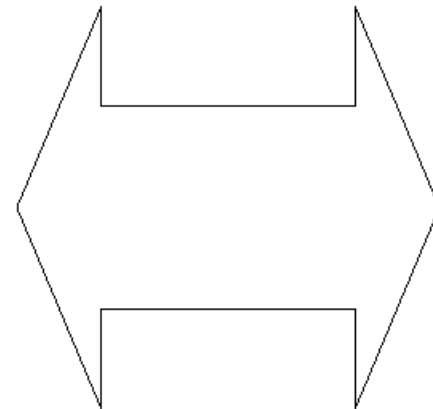
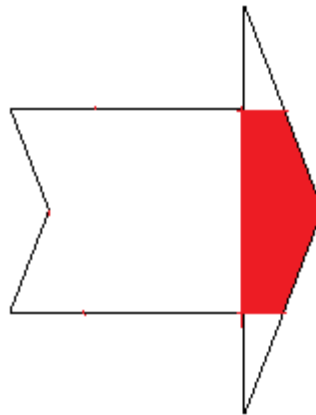
# Полигонизация

- Конечный набор точек на плоскости может быть различным образом соединён рёбрами для образования полигона.
- Формирование каждого такого полигона можно назвать *полигонизацией* набора точек.



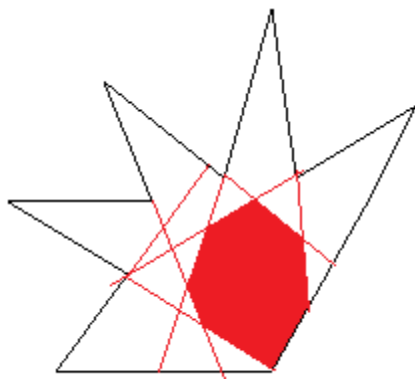
# Звездчатый полигон

- *Ядро* полигона – множество точек, из которых видны все вершины полигона. Эта точка принадлежит либо полигону, либо его границе. «Видно» точку - это значит, что из нее можно провести хорду во все вершины полигона.
- *Звездчатый* полигон – полигон, у которого ядро не пусто.



# Веерообразный полигон

- *Веерообразный* полигон – это полигон, у которого хотя бы одна вершина принадлежит его ядру. Каждая такая вершина называется *корнем* полигона.
- Любой выпуклый полигон является веерообразным, так как его ядро включает все вершины.
- Любой веерообразный полигон является звёздчатым, так как его ядро непустое.





# Виды полигонов

Выпуклый → Веерообразный → Звёздчатый

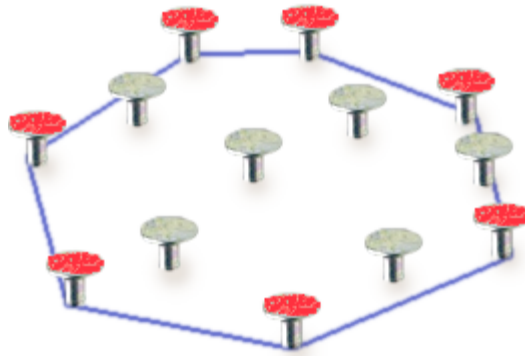
Незвёздчатый

# План

- Виды полигонов
- **Выпуклые оболочки**
- Триангуляция
- Пересечение/объединение

# Выпуклые оболочки

- Пусть  $S$  будет конечным набором точек на плоскости.
- Выпуклая оболочка набора  $S$ , обозначаемая как  $CH(S)$ , равна пересечению всех выпуклых полигонов, которые содержат  $S$ .
- Иными словами,  $CH(S)$  – это выпуклый полигон минимальной площади, содержащий все точки  $S$ .
- Существует еще одно определение, которое утверждает, что  $CH(S)$  равно объединению всех треугольников, определяемых точками набора  $S$ .



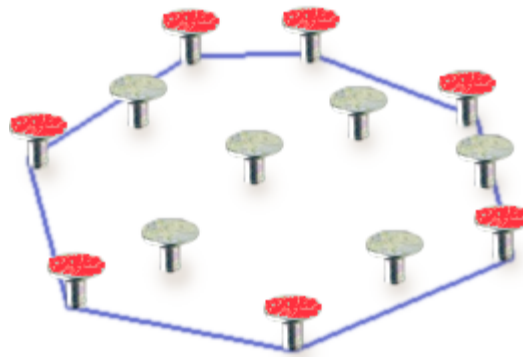
# Алгоритмы построения выпуклых оболочек

- Джарвиса (Jarvis march) – заворачивания подарка (Gift wrapping algorithm )
- Грехема (Graham scan)
- Эндрю (Andrew)
- Чана (Chan's algorithm)
- QuickHull
- Добавление точек пошагово

# Алгоритм Джарвиса (заворачивания подарка)

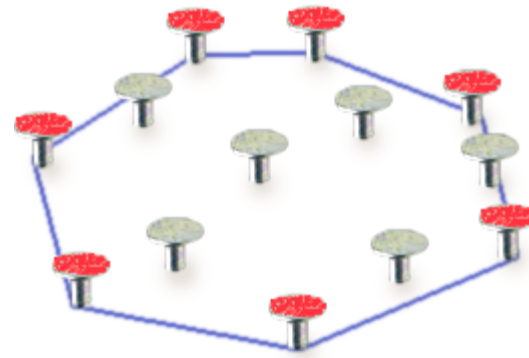
R. A. Jarvis, 1973

- можно представить как обтягивание верёвкой множества вбитых в доску гвоздей

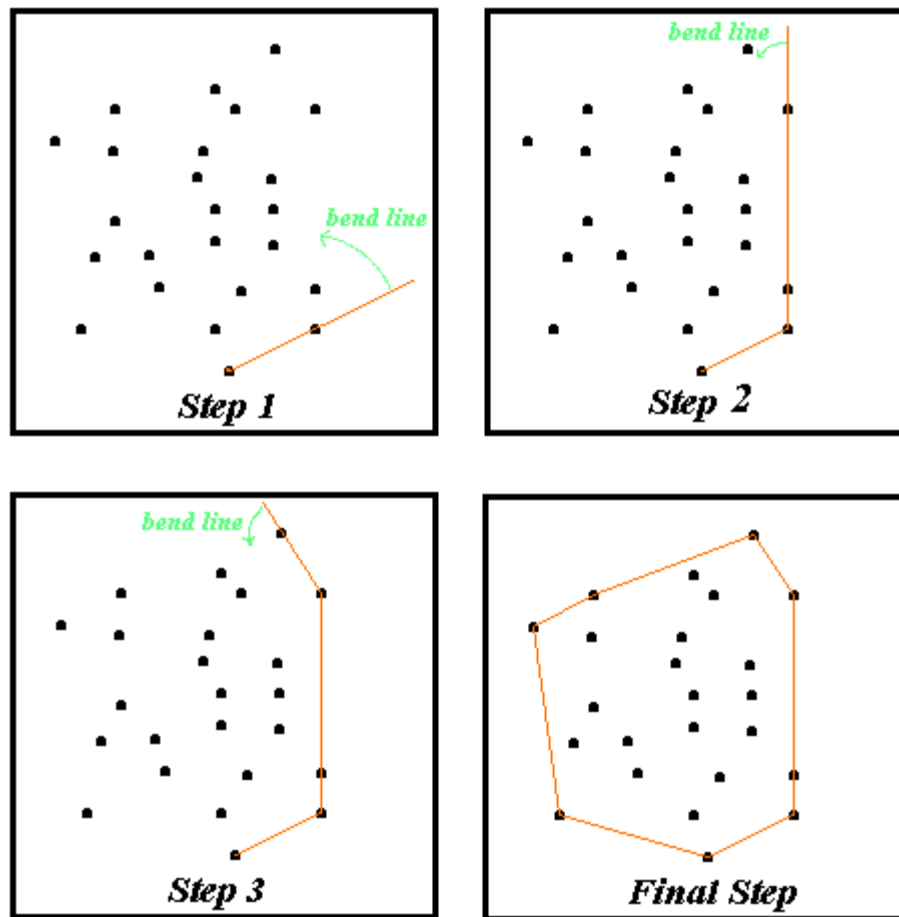


# Алгоритм Джарвиса (заворачивания подарка)

- Вначале выбирается некоторая точка  $S_0$  такая, что она явно должна принадлежать границе выпуклой оболочки.
  - Для этого годится самая левая точка. Если их несколько, то самая нижняя точка из них.
  - Можно самая правая.
  - Можно самая нижняя или самая верхняя))))

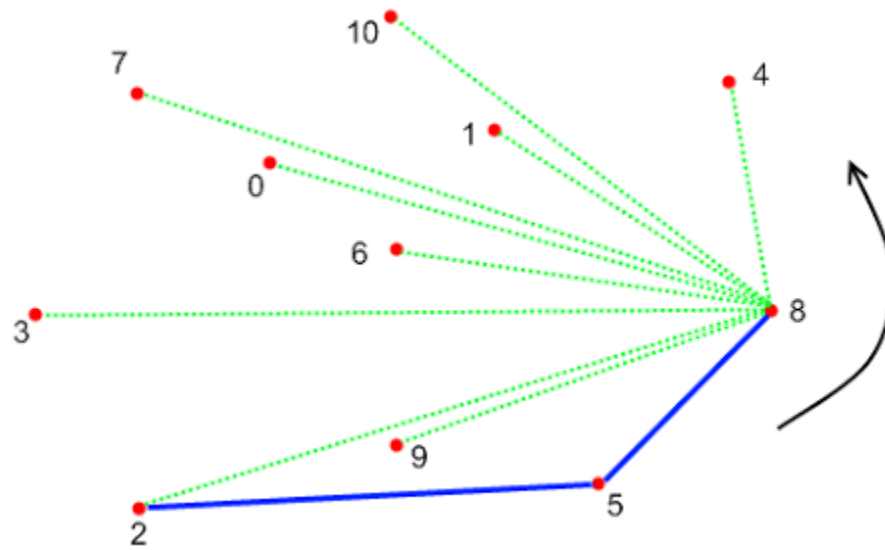


# Алгоритм Джарвиса (заворачивания подарка)



# Алгоритм Джарвиса (заворачивания подарка)

- на каждом следующем шаге для последней добавленной точки ищем среди всех недобавленных точку с максимальным углом (минимальным косинусом  $\rightarrow$  скалярным произведением)
- или самую правую (левую) точку





# Сложность алгоритм Джарвиса (заворачивания подарка)

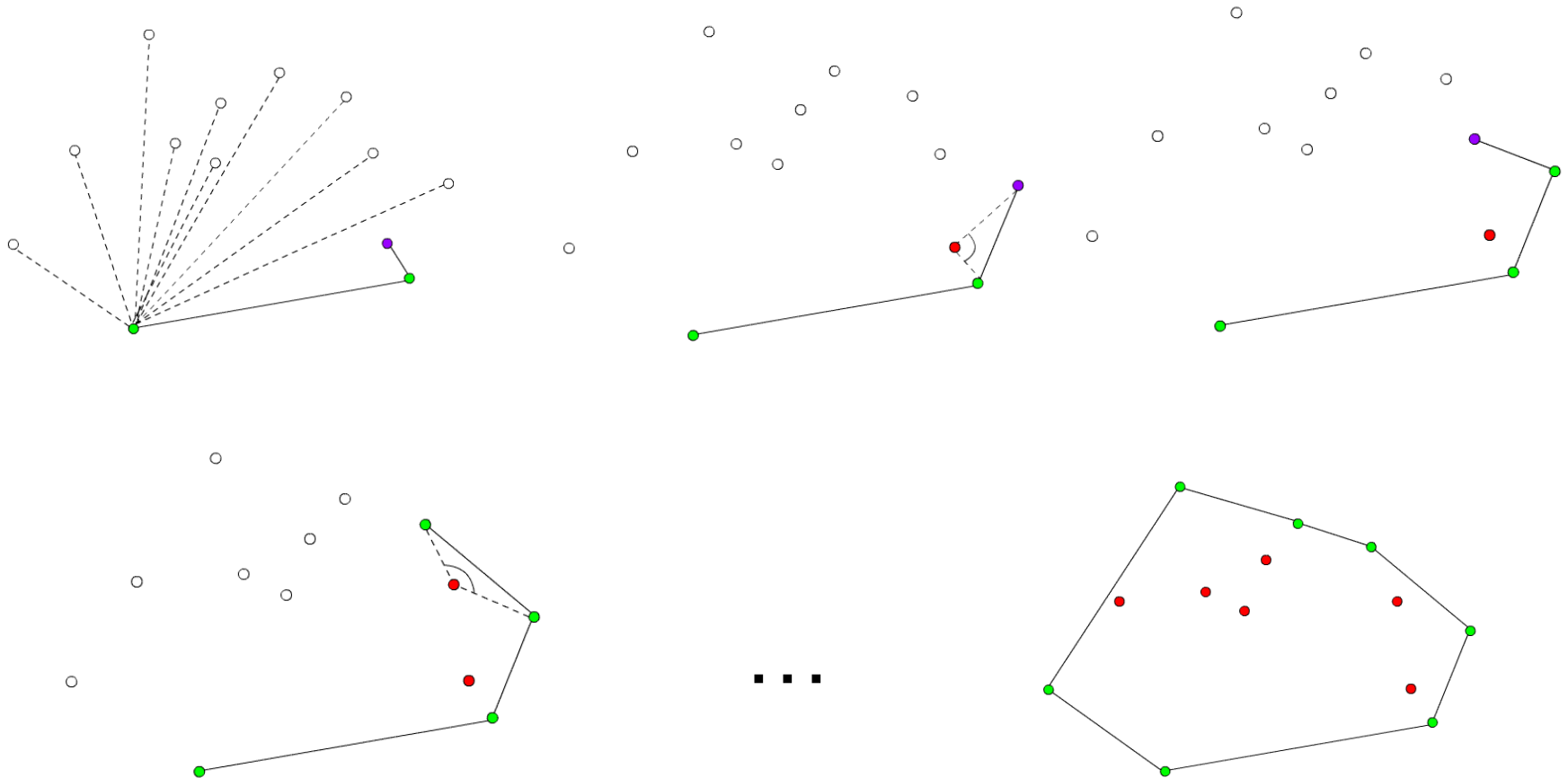
- $O(hn)$ , где  $n$  - количество всех вершин и  $h$  - количество вершин попавших в выпуклую оболочку.

# Алгоритмы построения выпуклых оболочек

- Джарвиса (Jarvis march) – заворачивания подарка (Gift wrapping algorithm )
- Грехема (Graham scan)
- Эндрю (Andrew)
- Чана (Chan's algorithm)
- QuickHull
- Добавление точек пошагово

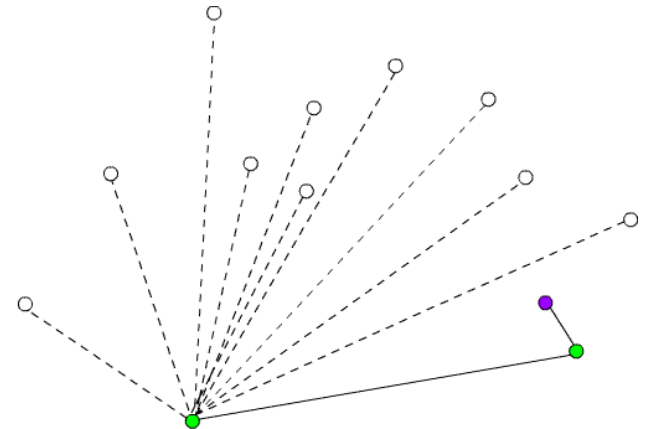
# Алгоритм Грехема

## Ronald Graham, 1972



# Алгоритм Грехема. Упорядочивание точек

- Выбирается экстремальная точка  $S_0$  (самая нижняя точка)
- Затем остальные точки сортируются в порядке увеличения полярного угла относительно точки  $S_0$ . Если полярные углы точек равны, то точка меньше, если ее радиус (расстояние до точки  $S_0$ ) меньше.



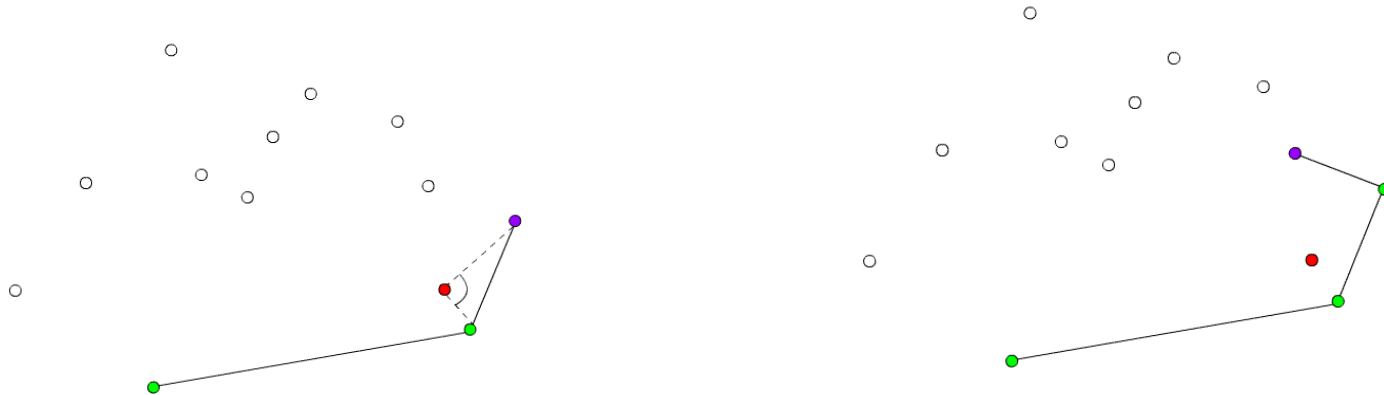
# Алгоритм Грехема. Обработка текущей точки

Обозначим:

$S_i$  – последняя занесенная точка;

$S_{i+1}$  – рассматриваемая текущая точка.

Рассмотрим  $S_{i+1}$ : если она находится не слева от луча  $S_{i-1}S_i$ , то удаляем из списка  $S_i$  - тую точку.



# Алгоритм Грехема.

## Обработка текущей точки

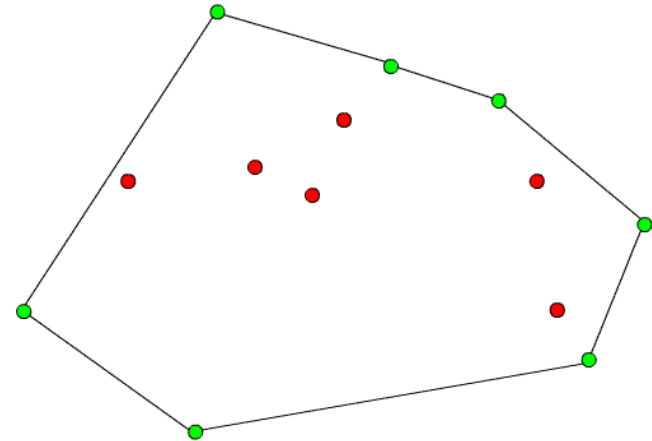
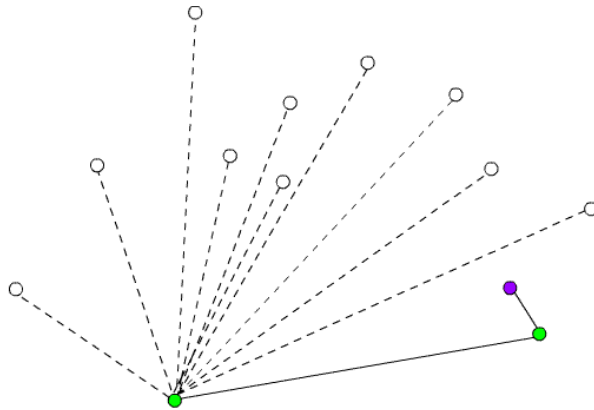
Далее опять проверяем, находится слева или не слева точка  $S_{i+1}$ , но только уже относительно луча  $S_{i-2}S_{i-1}$ .

Повторяем проверки и исключения из списка до тех пор, пока:

- 1) либо точка  $S_{i+1}$  не станет слева от рассматриваемого луча;
- 2) либо не останется в списке только одна точка, не считая  $S_{i+1}$ .

Затем добавляем точку  $S_{i+1}$  в конец списка.

# Алгоритм Грехема. Время работы



Общая асимптотика  $O(n \lg n)$ , где  $n$  – количество точек.

Добавление/удаление потребует время  $O(n)$ .

Сортировка полярных углов займет  $O(n \lg n)$  времени.

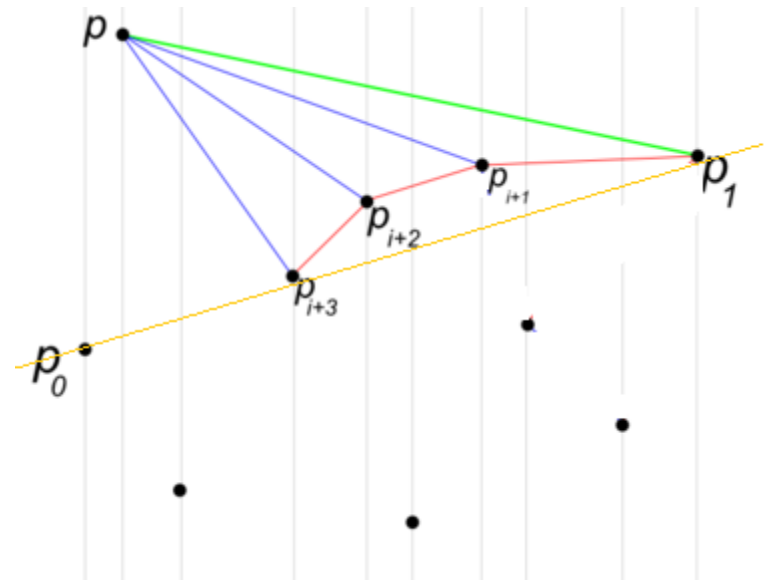
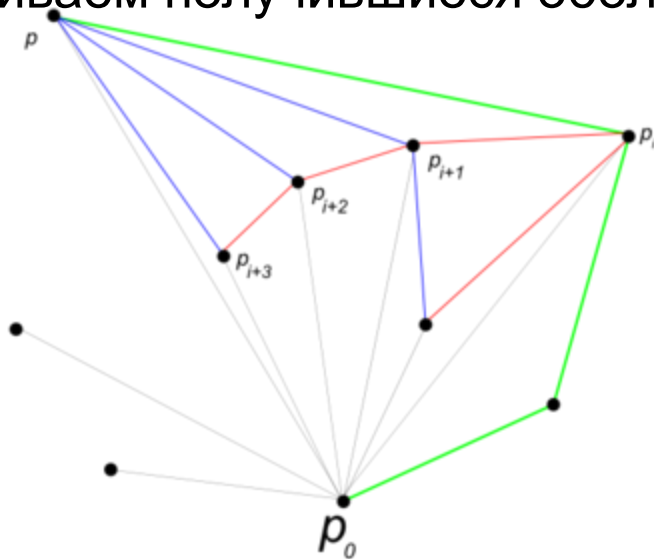
# Алгоритмы построения выпуклых оболочек

- Джарвиса (Jarvis march) – заворачивания подарка (Gift wrapping algorithm )
- Грехема (Graham scan)
- Эндрю (Andrew)
- Чана (Chan's algorithm)
- QuickHull
  
- Добавление точек пошагово



# Алгоритм Эндрю (Andrew) 1979 г

- Находим самую левую и самую правую точки множества.
- Делим множество на две части: точки над и под прямой.
- Для каждой половины ищем выпуклую оболочку алгоритмом Грехема с условием, что сортируем не по полярному углу, а по координате.
- Сливаем получившиеся оболочки.



# Алгоритмы построения выпуклых оболочек

- Джарвиса (Jarvis march) – заворачивания подарка (Gift wrapping algorithm )
- Грехема (Graham scan)
- Эндрю (Andrew)
- Чана (Chan's algorithm)
- QuickHull
- Добавление точек пошагово

# Алгоритм Чана (Chan's algorithm)

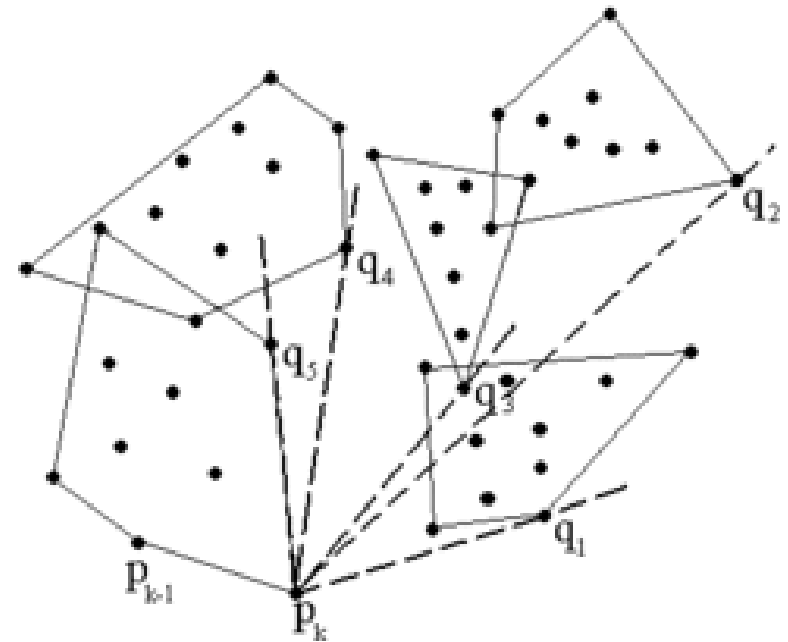
Тимоти М. Чан, 1996

Является комбинацией двух более медленных алгоритмов (Грехем и Джарвис).

Грехем –  $O(n \cdot \log n)$

Джарвис –  $O(hn) \rightarrow O(n^2)$ , .

- Разобьем всё множество на произвольные группы по  $m$  штук в каждой.
- Для каждой группы запускаем Грехема.
- Начиная с самой нижней точки ищем самую выпуклую оболочку Джарвисом, но перебираем не все точки, а только те, что вошли в оболочки групп.



# Алгоритм Чана (Chan's algorithm)

Тимоти М. Чан, 1996

Является комбинацией двух более медленных алгоритмов (Грэхем и Джарвис).

Количество групп  $r = \lceil n/m \rceil$

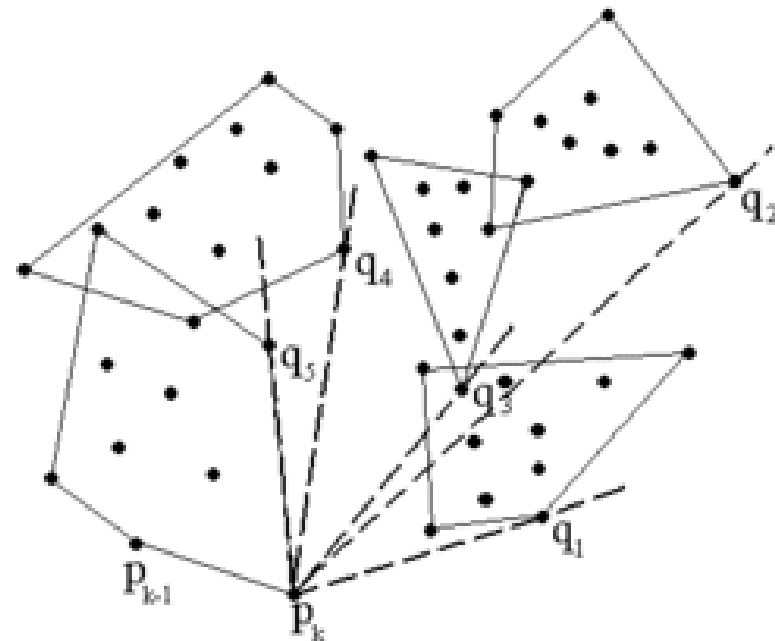
Грэхем для группы  $O(m \cdot \log m)$

Грэхем для всех групп  $O(r \cdot m \cdot \log m) = O(n \cdot \log m)$

Джарвис  $O(h \cdot r \cdot \log m) = O(h \cdot n/m \cdot \log m)$

Чан  $O((n + h \cdot n/m) \cdot \log m)$

Если  $m \geq h$ , то Чан  $O(n \cdot \log m)$



# Выбор разбиения

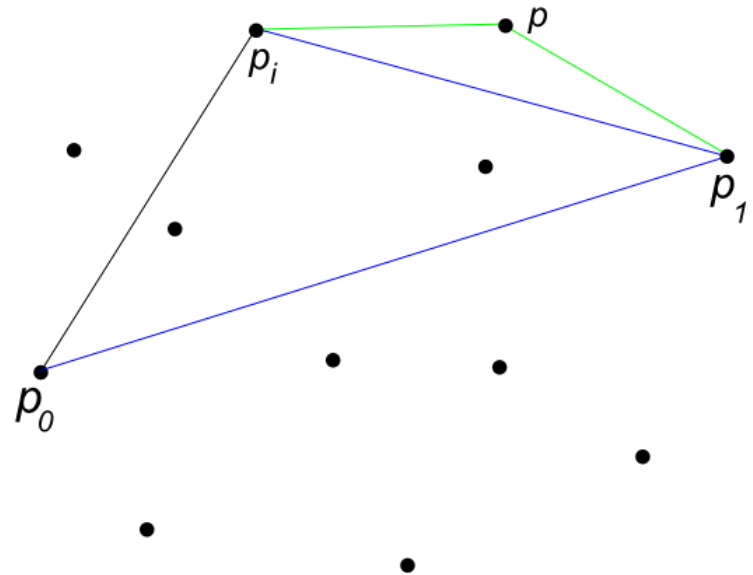
```
ChanHull(P)
  for t = 1 to n do:
    (a) взять  $m = \min(2^{2^t}, n)$ ;
    (b) L = Hull(P, m);
    (c) if L != «m маленькое, попробуйте еще» return L;
```

# Алгоритмы построения выпуклых оболочек

- Джарвиса (Jarvis march) – заворачивания подарка (Gift wrapping algorithm )
- Грехема (Graham scan)
- Эндрю (Andrew)
- Чана (Chan's algorithm)
- QuickHull
- Добавление точек пошагово

# Алгоритм QuickHull

- Найдем самую левую и самую правую точки.
- Возьмем все точки выше прямой, их соединяющей.
- Найдем точку, наиболее отдаленную от прямой (если таких несколько, взять самую правую).
- Рекурсивно повторить шаги 2-3 пока есть точки.
- Добавить в ответ все точки.
- Повторить пункты 2-5 для для "нижней" половины.



# Алгоритм QuickHull. Сложность

- Лучший случай, все точки внутри оболочки:  $O(n)$
- Худший случай, все точки на оболочке:  $O(n^2)$
- Среднее время:  $O(n \log n)$
- На случайном наборе точек этот алгоритм работает быстрее всех.



# Алгоритм QuickHull. Обобщение

- Алгоритм QuickHull может быть естественным образом обобщен на случай произвольной размерности.
- Трехмерный случай  
<http://algotist.manual.ru/maths/geom/convhull/qhull3d.php>
- Общий случай произвольной размерности в оригинальной работе Барбера  
[http://algotist.manual.ru/maths/geom/convhull/qhull\\_96.zip](http://algotist.manual.ru/maths/geom/convhull/qhull_96.zip)

# Алгоритмы построения выпуклых оболочек

- Джарвиса (Jarvis march) – заворачивания подарка (Gift wrapping algorithm )
- Грэхема (Graham scan)
- Эндрю (Andrew)
- Чана (Chan's algorithm)
- QuickHull
- Добавление точек пошагово (Выпуклая оболочка онлайн)

# Добавление точек пошагово

## Выпуклая оболочка онлайн

- Алгоритмы выше принимают на вход все множество точек сразу.
- А если точки подаются одна за другой, а задача состоит в том, чтобы поддерживать выпуклую оболочку имеющихся точек?

# Добавление точек пошагово

## Выпуклая оболочка онлайн

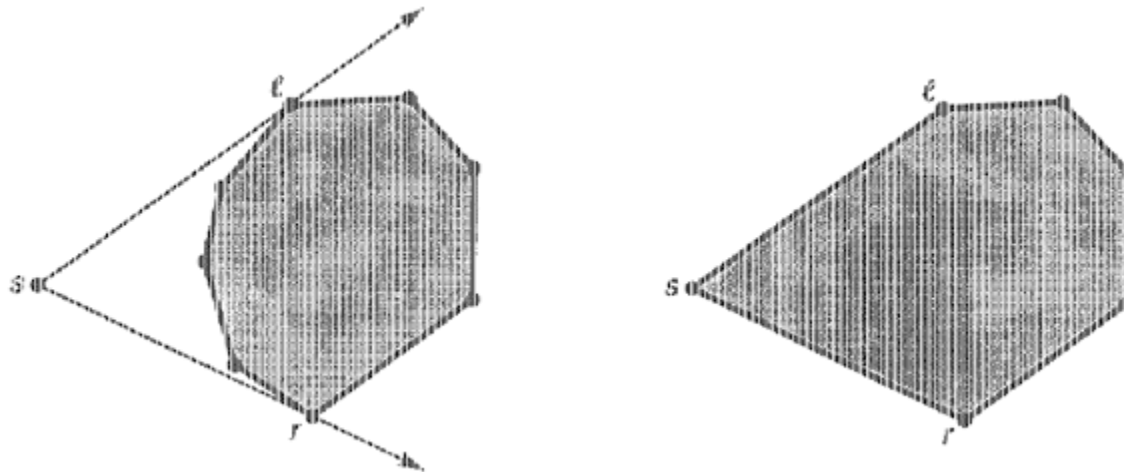
- Самый простой выход – после каждой добавленной точки вызывать какой-нибудь из описанных выше алгоритмов.
- Он построит выпуклую оболочку за  $O(n \log n)$  операций. Это приведет к сложности  $O(n^2 \log n)$ .
- Если хочется сделать побыстрее и поизящнее, то можно применить специальный алгоритм, работающий за  $O(n^2)$ .

# Добавление точек пошагово

## Выпуклая оболочка онлайн

Вначале текущая оболочка состоит из одной точки, принадлежащей  $S$ .

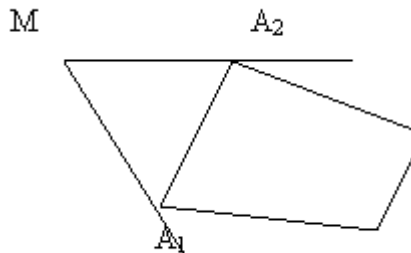
По завершении ввода всех точек текущая оболочка становится эквивалентной выпуклой оболочке  $CH(S)$  и задача оказывается решенной.



Ласло "Вычислительная геометрия и компьютерная графика на C++"

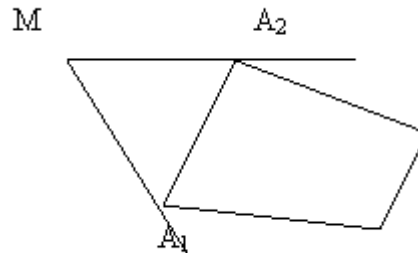
# Добавление точек пошагово

- Пусть три точки уже есть. Строим треугольник. Далее в цикле добавляем новые точки. При вводе каждой новой точки  $M$  в текущую оболочку могут возникнуть две ситуации.
- В первом случае точка  $M$  может принадлежать текущей оболочке (находится на ее границе или внутри) и тогда текущую оболочку изменять не требуется.
- Во втором случае, если точка снаружи, то необходимо модифицировать текущую оболочку.



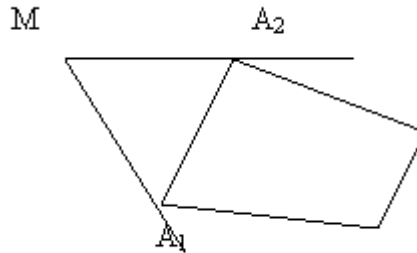
# Добавление точек пошагово. Поиск касательных

- Ищем левую и правую касательные из исходной точки к данному полигону.
- Левая касательная – это луч через такую точку, для которой все существующие точки полигона находятся справа от этого луча.
- Соответственно, правая касательная – это луч через такую точку, для которой все существующие точки полигона находятся слева от этого луча.



# Добавление точек пошагово. Ближняя и дальняя цепочки

- Ближняя цепочка расположена ближе к точке М, дальняя цепочка находится дальше от точки М.
- Изъять всю ближнюю цепочку от правой до левой касательной (за исключением вершин  $A_2$  и  $A_1$ ).
- Добавить в список между  $A_1$  и  $A_2$  точку М.
- Если все точки лежат на одной прямой (т.е. касательных нет), М просто добавляется в список, а  $A_1$  удаляется.





# План

- Виды полигонов
- Выпуклые оболочки
- **Триангуляция**
- Пересечение/объединение

# Разбиение (триангуляция) полигонов

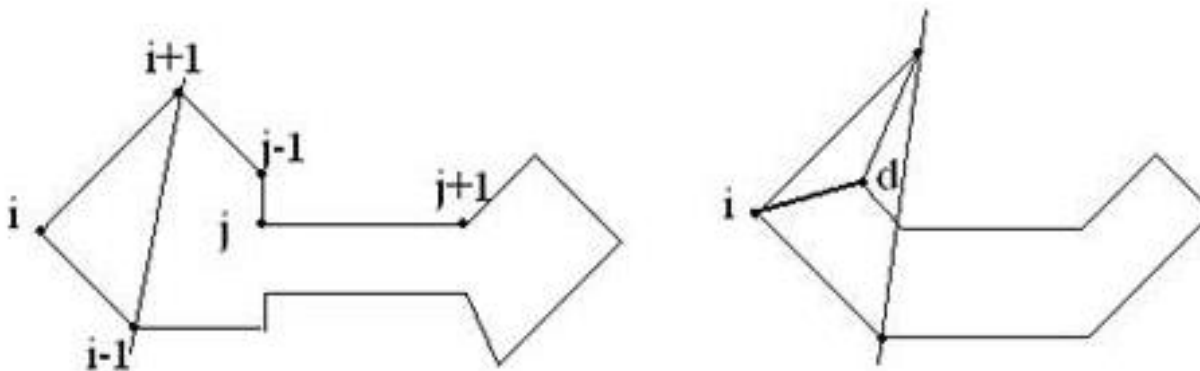
- *Триангуляцией* полигона называется декомпозиция полигона в набор треугольников.

# Триангуляция

- Триангуляция полигона методом вторгающейся вершины
- Триангуляция монотонного полигона
- Триангуляция Делоне для конечного набора точек

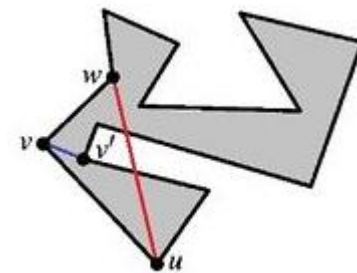
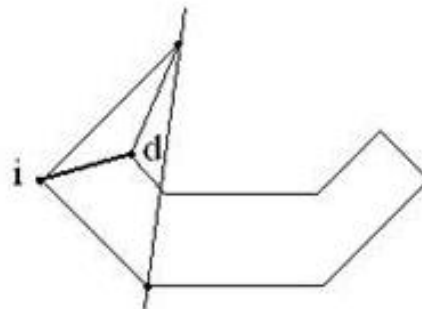
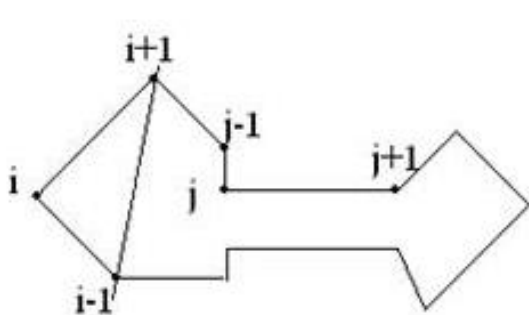
# Триангуляция полигона методом вторгающейся вершины

- Берем любую вершину и ищем ближайшую к ней выпуклую вершину.
- Если  $(i+1)$  точка лежит справа от ребра  $(i-1, i)$ , то точка выпуклая, иначе вогнутая.
- Ищем вторгающиеся вершины в треугольнике  $i-1, i, i+1$ . Из них выбираем наиболее удаленную от прямой  $i-1, i+1$ .



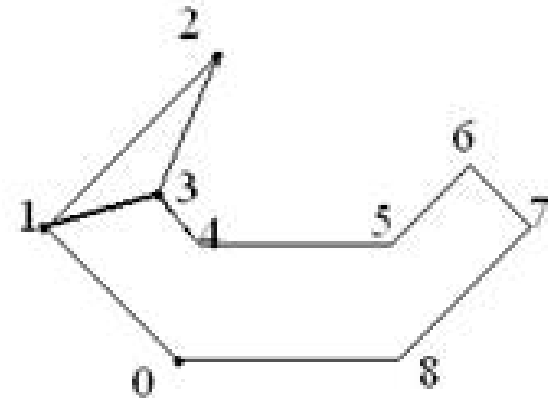
# Триангуляция полигона методом вторгающейся вершины

- Если вторгающихся вершин не обнаружено, то отсекаем треугольник  $i-1, i, i+1$ , добавляем его в список треугольников, а из полигона удаляется  $i$ -тая вершина.
- Иначе производим разбиение полигона на два ребром с вершинами в точке  $i$  и найденной вторгающейся вершиной.
- Повторяем разбиение для полученных полигонов рекурсивно.
- В первом случае полигон 1, во втором – 2. Повторяем, пока полигон не превратится в треугольник.



# Как разрезать полигон на 2?

Разбивание по ребру: дублируем вершины, разрезаем его и переопределяем связи:



# Триангуляция монотонного полигона

- Монотонный полигон образуется из 2 цепочек: верхней монотонной цепочки и нижней монотонной цепочки.
- Цепочка ребер называется монотонной, если ни один перпендикуляр к прямой между началом и концом цепочки не пересекает цепочку более одного раза.



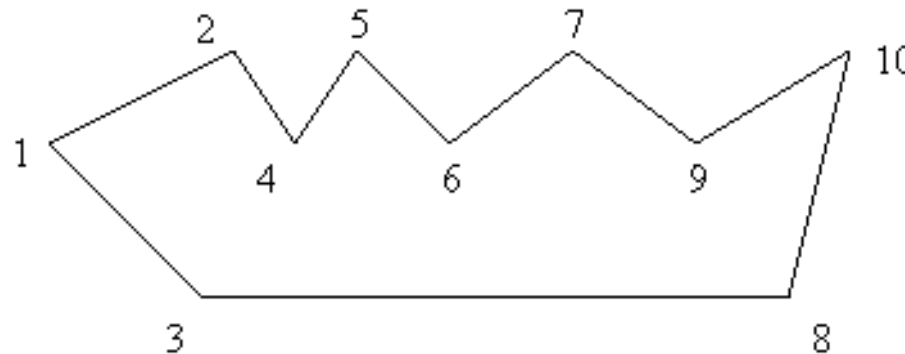
(a)



(б)

# Триангуляция монотонного полигона

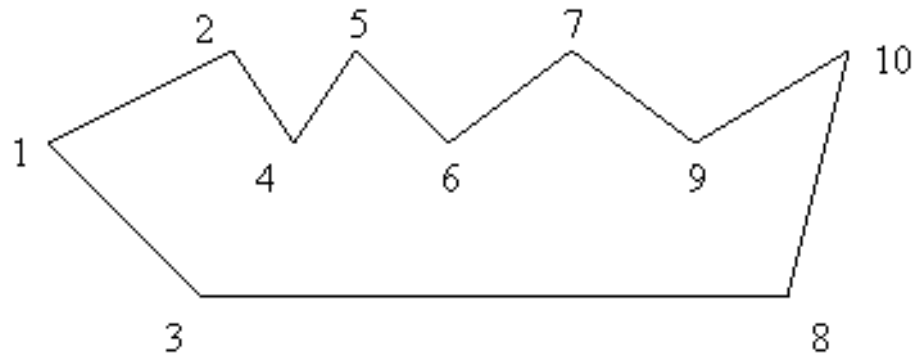
- Будем использовать стек, содержащий вершины. На выходе должны получить список треугольников.
- Вначале найдем левый и правый концы. Все, что по часовой стрелке слева направо – верхняя цепочка, а справа налево – нижняя.
- Необходимо также пометить в списке, какой цепочке принадлежит вершина.
- Нужно получить список всех вершин отсортированных по оси  $Ox$ .





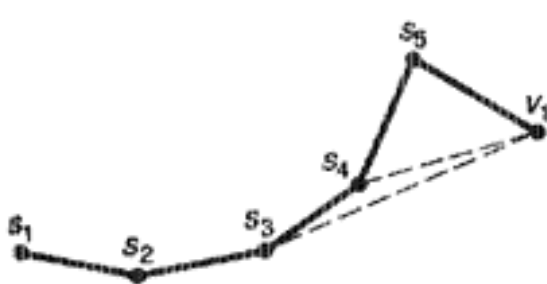
# Триангуляция монотонного полигона

- В стек заносим левую точку.
- Берем следующую точку из отсортированных вершин.
- Если в стеке только одна вершина, то просто добавляем ее в стек.
- Если в стеке больше вершин, то проверяем, не образуют ли они треугольник или веерообразный полигон.
- **!!!** В нашем стеке первая вершина всегда доступна для просмотра.

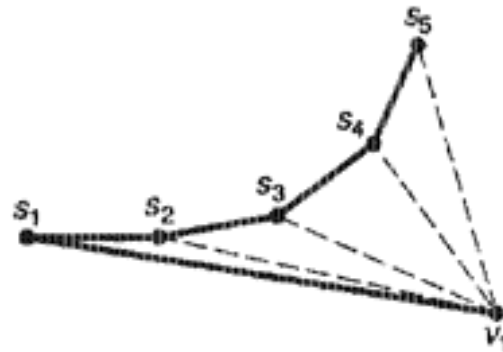


# Триангуляция монотонного полигона

- $S_t$  – последняя вершина в стеке.
- $S_1$  – первая вершина в стеке.
- При добавлении  $i$ -той вершины могут возникнуть 4 ситуации:
  - если  $V_i$  соседняя с  $S_t$  и не соседняя с  $V_1$ , то в случае, если внутренний угол  $S_{t-1}S_tV_i$  меньше 180 градусов, то мы отсекаем треугольник  $S_{t-1}S_tV_i$ , т.е. в список треугольников заносим этот треугольник, в полигоне удаляем эту вершину, из стека удаляем  $S_t$  и опять анализируем  $V_i$ .



(a)



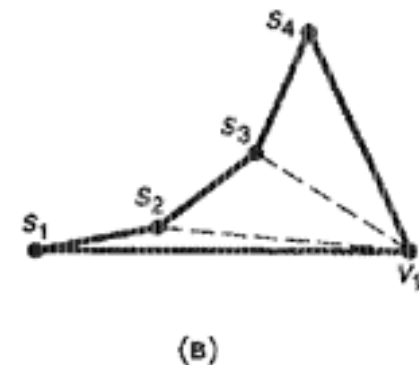
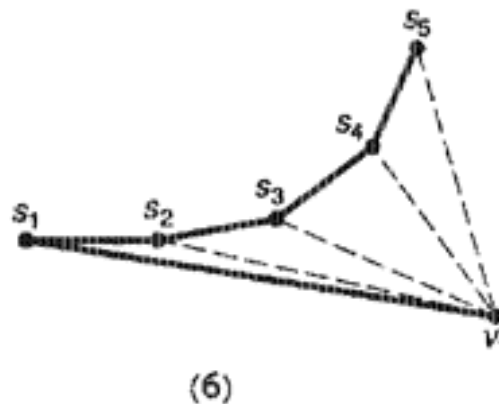
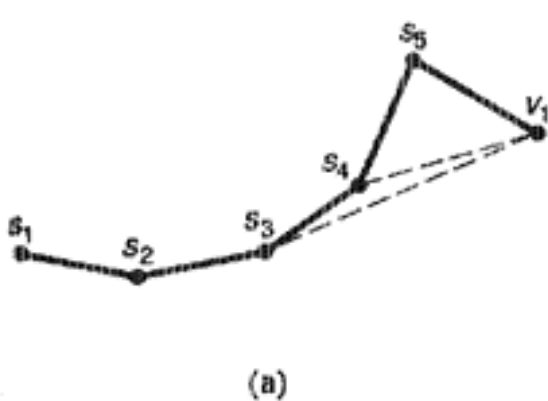
(б)

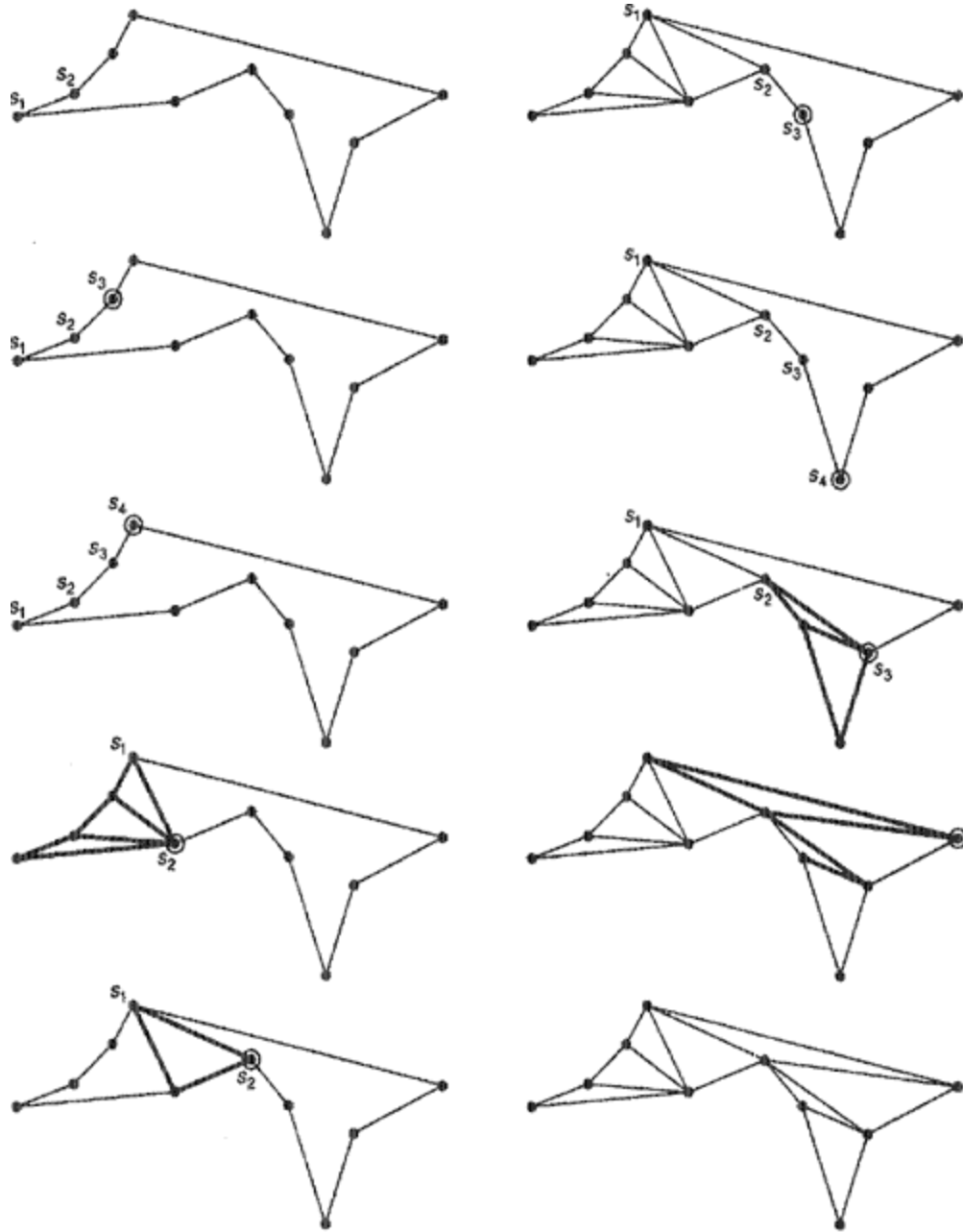


(в)

# Триангуляция монотонного полигона

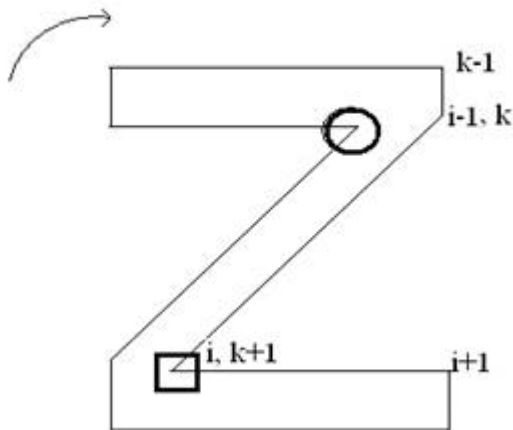
- Если  $V_i$  – соседняя с  $S_1$ , но не соседняя с  $S_t$ , тогда получим веерообразный полигон.
- Отсекаем и бьем на треугольники, заносим их в список треугольников.
- Из стека удаляем все вершины и добавляем в него  $S_t V_i$  и рассматриваем следующую  $V_i$ .
- Если  $V_i$  – соседняя с  $S_1$  и с  $S_t$ . Это означает, что  $V_i$  совпадает с  $V_n$  (последняя рассматриваемая точка) и получаем веерообразный полигон. Бьем его на треугольники и добавляем в список треугольников.
- Если ни одна из предыдущих ситуаций не подходит, то просто добавляем  $V_i$  в стек.





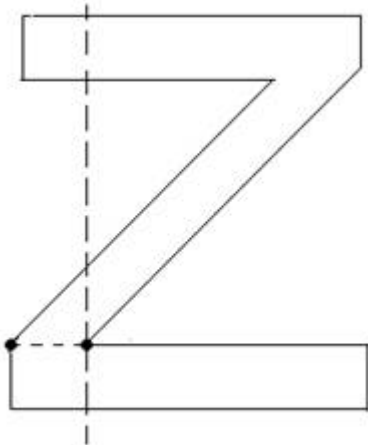
# Разбиение немонотонного на монотонные

- Упорядочиваем все вершины по оси  $Ox$ . Выбираем самую левую вершину полигона.
- Начинаем двигаться слева направо до первого изгиба влево. В немонотонном полигоне монотонность нарушают изгибы.
- Они делятся на изгибы влево (*на рис. квадратик*) и изгибы вправо (*на рис. кружочек*). Изгиб является изгибом влево, если вершины  $(i+1)$ ,  $(i-1)$  имеют  $x$  больший ( $\geq$ ), чем вершина  $i$ . И соответственно, если меньший, то изгиб вправо.
- При таком определении существует опасность: можно внешнюю вершину принять за изгиб. Нас интересуют внутренние углы: если он больше  $180$  градусов, то этот угол является изгибом.



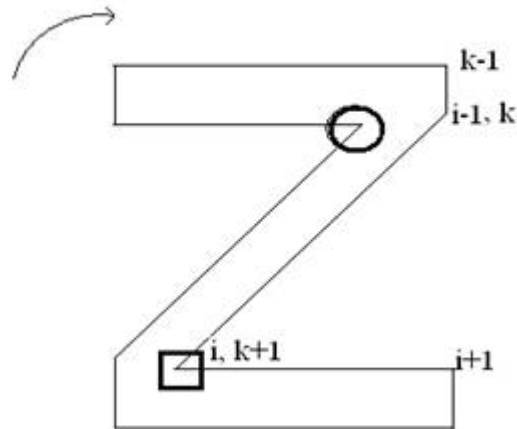
# Разбиение немонотонного на монотонные

- производим разбиение по первому изгибу влево:
  - а) через вершину изгиба проводим вертикальную прямую, чтобы найти ближайшие к вершине пересечения с верхним и нижним ребром.
  - б) у этих ребер выбираем ближайшую по  $x$  вершину слева. Ищем между  $k$ -той и  $r$ -той вершиной ближайшую к  $r$ -той, если нет, то берем  $k$ -тую и разбиваем весь полигон на два.



# Разбиение немонотонного на монотонные

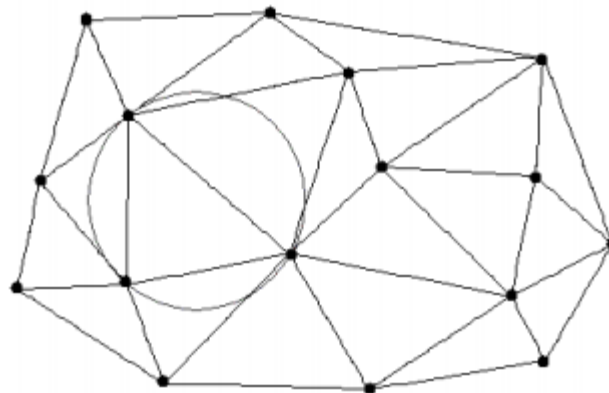
- продолжаем по отсортированному списку вершин двигаться вправо. Если находим очередной изгиб влево, то опять разбиваем.
- двигаемся справа налево в поисках изгиба вправо (для каждого полигона). Разбиение изгиба вправо производится аналогичным образом, только ближайшие точки ищутся справа.



# Триангуляция Делоне

советский математик Борис Делоне, 1934

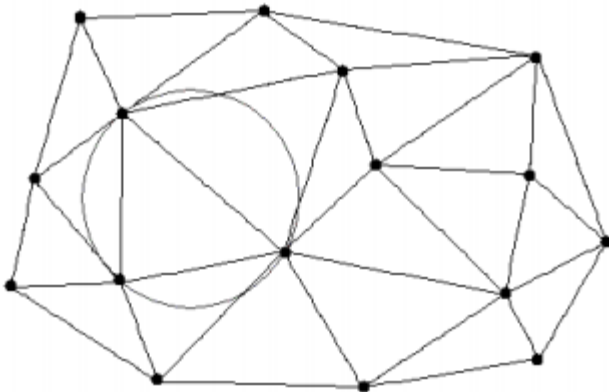
- *Триангуляция для конечного набора точек  $S$  является задачей триангуляции выпуклой оболочки  $CH(S)$ , охватывающей все точки набора  $S$ .*
- Отрезки прямых линий при триангуляции не могут пересекаться – они могут только встречаться в общих точках принадлежащих ребру.





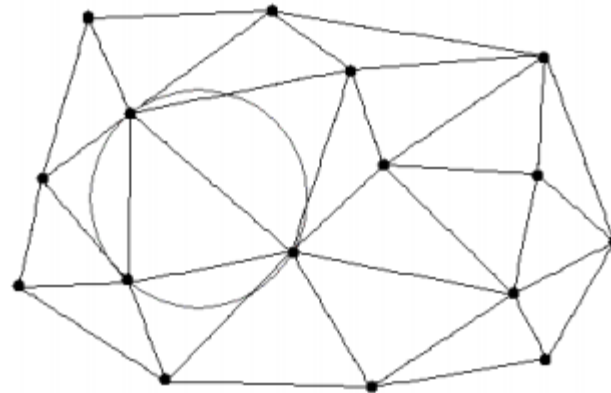
# Триангуляция Делоне

- Все точки из набора  $S$  могут быть подразделены на *граничные* точки – те точки, которые лежат на границе выпуклой оболочки  $CH(S)$ , и *внутренние* точки – лежащие внутри выпуклой оболочки  $CH(S)$ .
- Также можно классифицировать и ребра, полученные в результате триангуляции  $S$ , как *ребра оболочки* и *внутренние* ребра.
- К ребрам оболочки относятся ребра, расположенные вдоль границы выпуклой оболочки  $CH(S)$ , а к внутренним ребрам – все остальные ребра, образующие сеть треугольников внутри выпуклой оболочки.



# Триангуляция Делоне

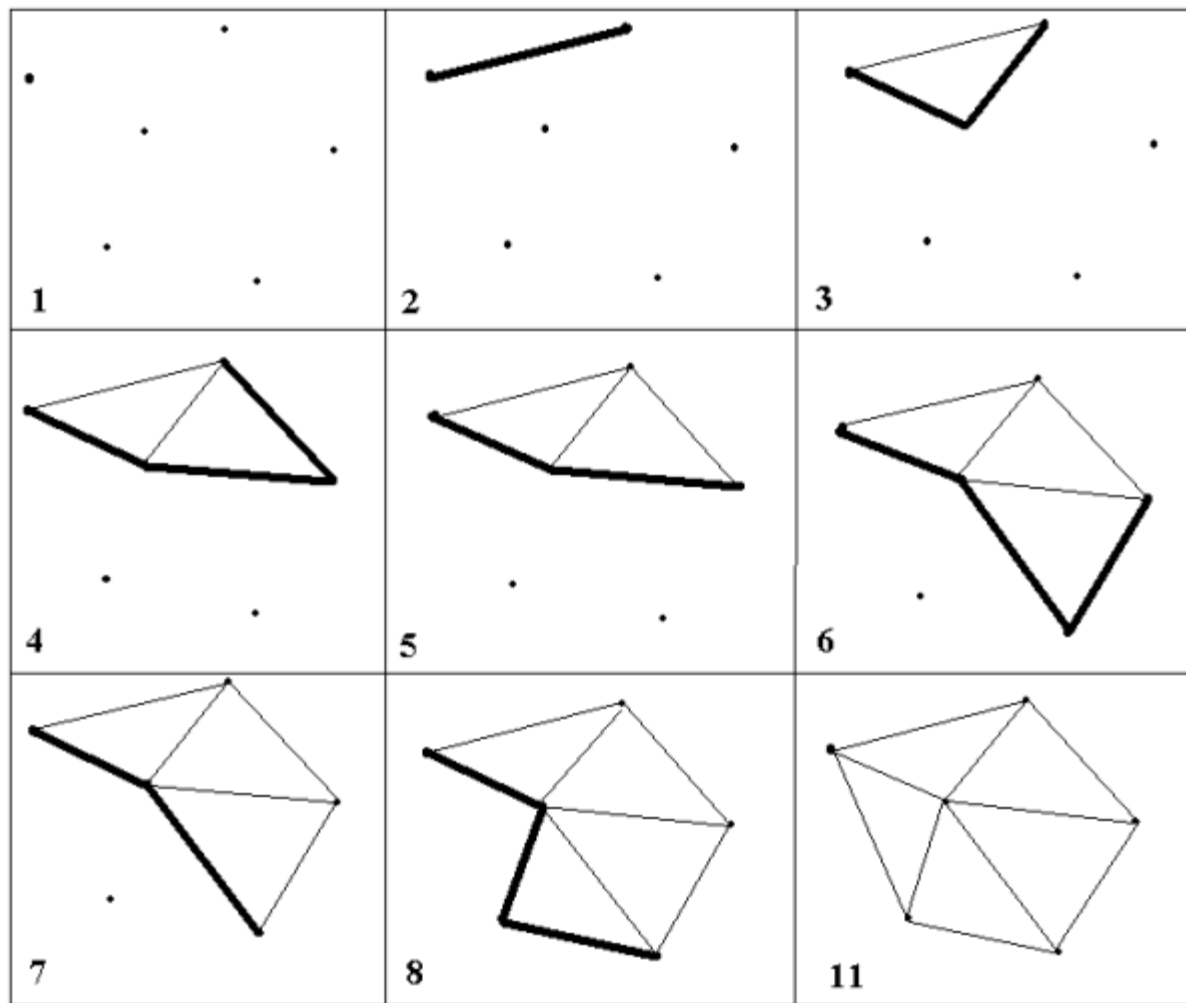
- *Триангуляция Делоне* хорошо сбалансирована в том смысле, что формируемые треугольники стремятся к *равноугольности*, т.е. по возможности максимально приближены к *равносторонним* треугольникам.



# Триангуляция Делоне

- Выбираем начальное ребро. Оно выбирается из алгоритма заворачивания подарков.
- Ребра, участвующие в построении полигона.
  - *Спящие* – ребра, которые далее будут рассматриваться.
  - *Мертвые* – уже рассматриваться не будут.
  - *Живые* – граница, не полностью обработанная.
- Итак, есть список точек, список треугольников, список живых ребер.
- Занесли первое ребро в список живых ребер таким образом, чтобы известная область была слева по направлению, а неизвестная – справа. Слева построен треугольник или бесконечность.

# Пошаговое добавление рёбер

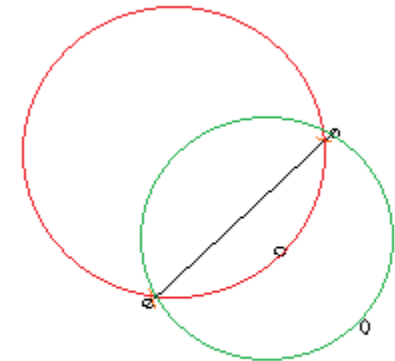


# Триангуляция Делоне

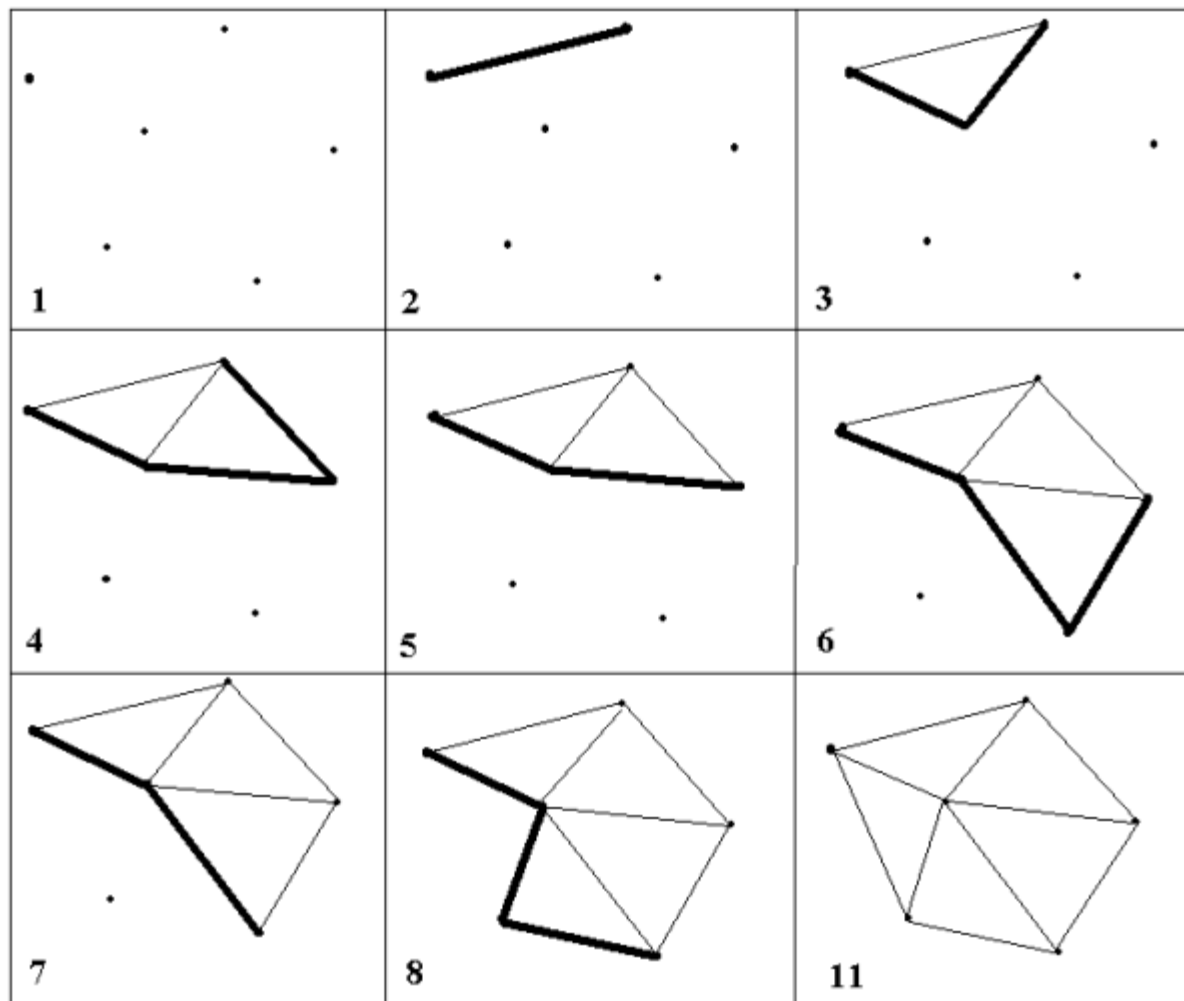
- Для каждого ребра из списка живых ребер: ищем правую сопряженную точку для построения треугольника.
- По правилу: правая сопряженная точка вместе с вершиной ребра должна образовывать окружность с минимальным расстоянием до центра окружности из всех возможных сопряженных точек справа. (Минимальное расстояние не по модулю, а по величине, с учетом знака). Если точка справа, а центр слева, то расстояние отрицательное.

• Известно, что центр описанной окружности находится в точке пересечения 2 серединных перпендикуляров.

• Если мы находим середину ребра параметрически, то все будет удачно, так как чем меньше  $t$ , тем меньше параметр.

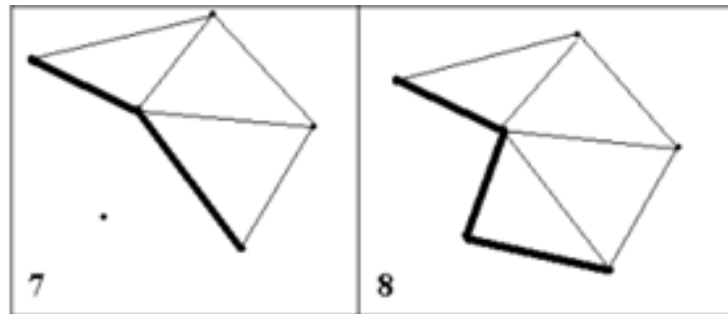


# Пошаговое добавление рёбер

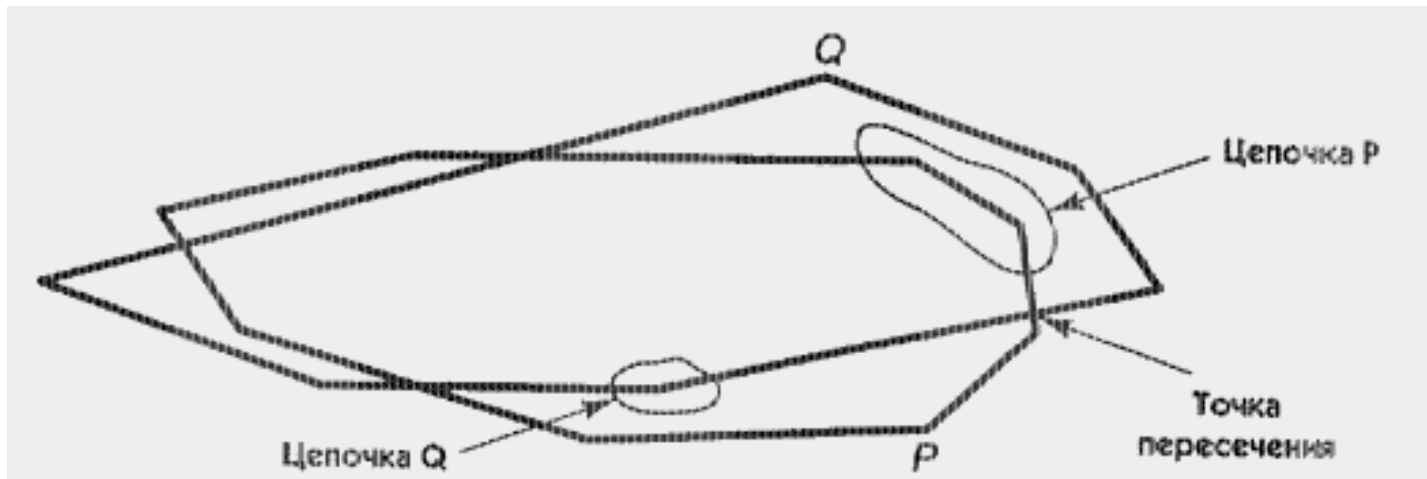


# Триангуляция Делоне

- Если вновь найденные ребра отсутствуют в списке живых ребер, то мы их заносим в список живых ребер, причем так, чтобы направление соответствовало известной левой области и неизвестной правой.
- Если ребро было в списке живых ребер, то мы его удаляем, и оно становится мертвым.
- Если при поиске сопряженной точки обнаружили, что нет ни одной точки справа, то это означает, что ребро принадлежит границе полигона (выпуклой оболочки), тогда новый треугольник не формируется, а ребро из списка живых ребер также удаляется.



# Пересечение и объединение выпуклых полигонов





# Литература

- *Прапарата Ф., Шеймос М.* Вычислительная геометрия: Введение = Computational Geometry An introduction. — М.: Мир, 1989. — С. 478.
- *Ласло М.* Вычислительная геометрия и компьютерная графика на C++. — М.: БИНОМ, 1997. — С. 304.
- *Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн.* Алгоритмы. Построение и анализ = Introduction to Algorithms. — 2-е изд. — “Вильямс”, 2005. — С. 1296.