

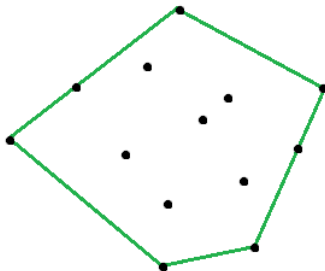
MIPT Fall Programming Training 2014

Выпуклые оболочки

Наталья Бондаренко

Саратовский государственный университет

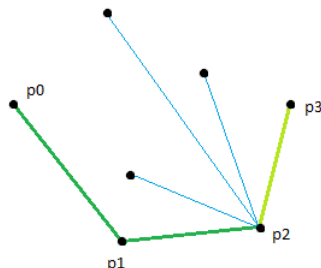
Выпуклой оболочкой заданного множества точек называется наименьшее выпуклое множество, содержащее заданное множество.



Выпуклая оболочка 2D

Алгоритм Джарвиса, или «заворачивание подарка»

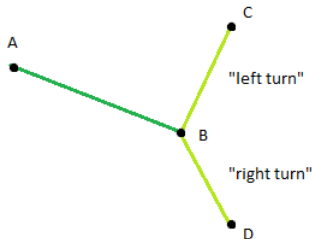
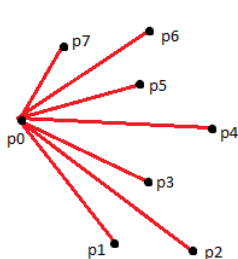
- 1 найдем самую левую точку p_0
- 2 на шаге i , выбираем точку p_{i+1} таким образом, чтобы все точки оказались слева от прямой $p_i p_{i+1}$



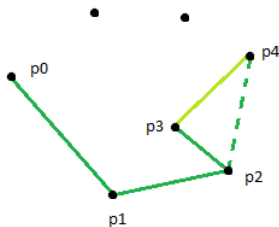
Вычислительная сложность алгоритма $O(nh)$, где n — количество заданных точек, а h — количество вершин выпуклой оболочки. $O(n^2)$ в худшем случае.

Обход Грэхема

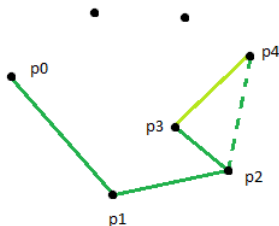
- 1 выберем самую левую точку p_0
- 2 отсортируем точки p_1, p_2, \dots, p_{n-1} по возрастанию углов, образуемых с осью x отрезками, соединяющими их с p_0
- 3 добавим точку p_1 к выпуклой оболочке
- 4 на каждом шаге определяем, «левый» или «правый» поворот имеет переход от отрезка между двумя последними добавленными точками выпуклой оболочки к новой точке в отсортированной последовательности



- 5 если «левый поворот», добавляем точку к выпуклой оболочке
- 6 если «правый поворот», удаляем последнюю точку из выпуклой оболочки и делаем так до тех пор, пока между последними двумя точками выпуклой оболочки и новой точкой сохраняется «правый поворот», затем добавляем новую точку к выпуклой оболочке



- 5 если «левый поворот», добавляем точку к выпуклой оболочке
- 6 если «правый поворот», удаляем последнюю точку из выпуклой оболочки и делаем так до тех пор, пока между последними двумя точками выпуклой оболочки и новой точкой сохраняется «правый поворот», затем добавляем новую точку к выпуклой оболочке



Вычислительная сложность $O(n \log n)$ из-за сортировки. Остальные шаги алгоритма выполняются за время $O(n)$.

Детали реализации

- если есть совпадающие точки, их нужно удалить

Детали реализации

- если есть совпадающие точки, их нужно удалить
- не забудьте обработать вырожденные случаи, когда выпуклая оболочка представляет собой точку или отрезок

Детали реализации

- если есть совпадающие точки, их нужно удалить
- не забудьте обработать вырожденные случаи, когда выпуклая оболочка представляет собой точку или отрезок
- нужно или не нужно включать точки, лежащие на ребрах выпуклой оболочки?

Детали реализации

- если есть совпадающие точки, их нужно удалить
- не забудьте обработать вырожденные случаи, когда выпуклая оболочка представляет собой точку или отрезок
- нужно или не нужно включать точки, лежащие на ребрах выпуклой оболочки?
- для сравнения полярных углов лучше всего использовать векторное произведение

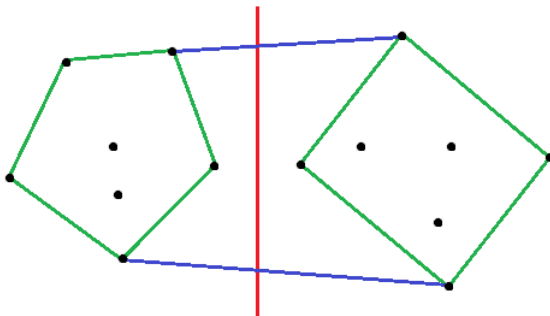
Детали реализации

- если есть совпадающие точки, их нужно удалить
- не забудьте обработать вырожденные случаи, когда выпуклая оболочка представляет собой точку или отрезок
- нужно или не нужно включать точки, лежащие на ребрах выпуклой оболочки?
- для сравнения полярных углов лучше всего использовать векторное произведение
- помните всегда, **дьявол в деталях**



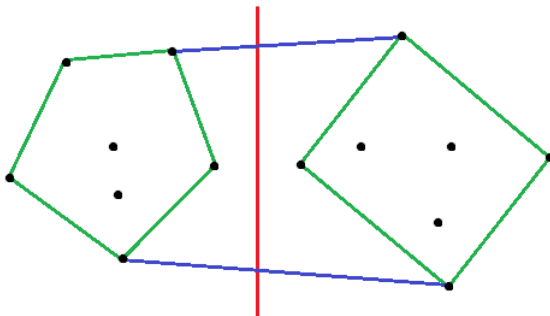
«Разделяй и властвуй»

- 1 отсортируем точки по координате x и разделим их на две равные части вертикальной прямой
- 2 построим выпуклую оболочку для обеих частей рекурсивно
- 3 найдем «МОСТЫ»



«Разделяй и властвуй»

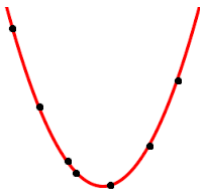
- 1 отсортируем точки по координате x и разделим их на две равные части вертикальной прямой
- 2 построим выпуклую оболочку для обеих частей рекурсивно
- 3 найдем «МОСТЫ»



Вычислительная сложность $O(n \log n)$.

Минимальная вычислительная сложность

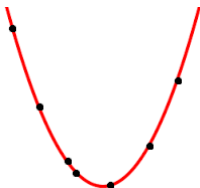
Рассмотрим точки на параболе, заданные в произвольном порядке.



В данном случае задача построения выпуклой оболочки эквивалентна задаче сортировки, поэтому наименьшее время, за которое она может быть решена — $O(n \log n)$.

Минимальная вычислительная сложность

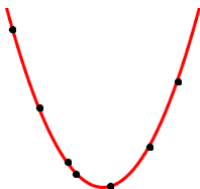
Рассмотрим точки на параболе, заданные в произвольном порядке.



В данном случае задача построения выпуклой оболочки эквивалентна задаче сортировки, поэтому наименьшее время, за которое она может быть решена — $O(n \log n)$. В некоторых случаях можно уменьшить время до $O(n)$, если использовать сортировку подсчетом или если точки отсортированы изначально.

Минимальная вычислительная сложность

Рассмотрим точки на параболе, заданные в произвольном порядке.



В данном случае задача построения выпуклой оболочки эквивалентна задаче сортировки, поэтому наименьшее время, за которое она может быть решена — $O(n \log n)$. В некоторых случаях можно уменьшить время до $O(n)$, если использовать сортировку подсчетом или если точки отсортированы изначально.

Существуют алгоритмы, вычислительная сложность которых зависит как от размера входных данных, так и от размера выходных данных (**output-sensitive** algorithms). С помощью таких алгоритмов построить выпуклую оболочку за $O(n \log h)$.

Алгоритм Киркпатрика-Зейделя

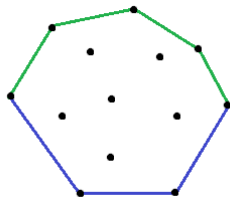
Kirkpatrick, David G.; Seidel, Raimund (1986). The ultimate planar convex hull algorithm. SIAM Journal on Computing 15 (1): 287–299.

Он напоминает алгоритм «разделяй и властвуй». Основная идея: «marriage-before-conquest» («женитьба до завоевания»). Сначала за $O(n)$ находятся «мосты», потом делается рекурсия.

Алгоритм Киркпатрика-Зейделя

Предположим, мы умеем находить мосты за время $O(n)$.

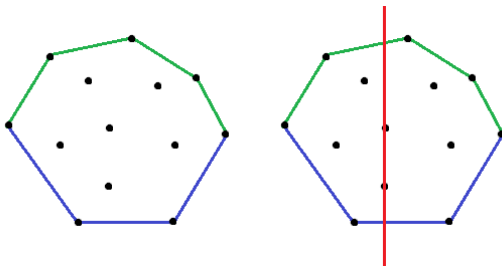
- 1 найдем самую левую и самую правую точки
- 2 отдельно построим верхнюю и нижнюю части оболочки



Алгоритм Киркпатрика-Зейделя

Предположим, мы умеем находить мосты за время $O(n)$.

- 1 найдем самую левую и самую правую точки
- 2 отдельно построим верхнюю и нижнюю части оболочки

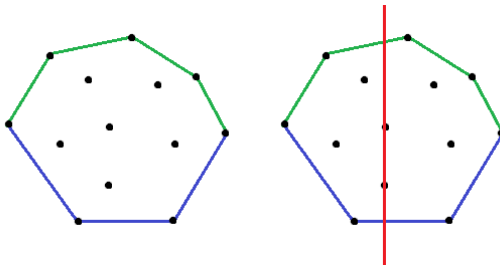


- 3 чтобы построить верхнюю часть, найдем среди точек медиану по координате x и проведем через нее вертикальную прямую

Алгоритм Киркпатрика-Зейделя

Предположим, мы умеем находить мосты за время $O(n)$.

- 1 найдем самую левую и самую правую точки
- 2 отдельно построим верхнюю и нижнюю части оболочки

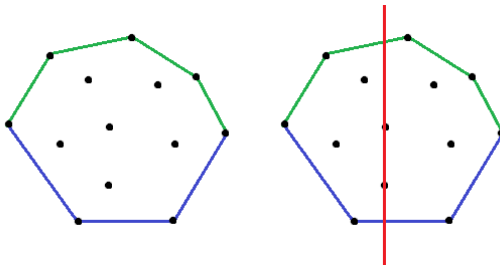


- 3 чтобы построить верхнюю часть, найдем среди точек медиану по координате x и проведем через нее вертикальную прямую
- 4 найдем «мост» через эту прямую

Алгоритм Киркпатрика-Зейделя

Предположим, мы умеем находить мосты за время $O(n)$.

- 1 найдем самую левую и самую правую точки
- 2 отдельно построим верхнюю и нижнюю части оболочки



- 3 чтобы построить верхнюю часть, найдем среди точек медиану по координате x и проведем через нее вертикальную прямую
- 4 найдем «мост» через эту прямую
- 5 удалим точки под «мостом» и построим левую и правую части верхней части оболочки рекурсивно

Алгоритм Киркпатрика-Зейделя

- на i -м уровне рекурсии будет 2^i подзадач
- каждая подзадача имеет размер не более $\frac{n}{2^i}$
- общее количество подзадач не превосходит h
- худший случай: полное бинарное дерево высоты $\log h$.

Общая вычислительная сложность алгоритма $O(n \log h)$.

Алгоритм Киркпатрика-Зейделя

Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре

Алгоритм Киркпатрика-Зейделя

Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре
- 2 если $x_1 = x_2$, исключим точку с меньшей координатой y

Алгоритм Киркпатрика-Зейделя

Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре
- 2 если $x_1 = x_2$, исключим точку с меньшей координатой y
- 3 для остальных пар посчитаем **наклоны**:

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

Алгоритм Киркпатрика-Зейделя

Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре
- 2 если $x_1 = x_2$, исключим точку с меньшей координатой y
- 3 для остальных пар посчитаем **наклоны**:

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

- 4 найдем медиану K множества наклонов k

Алгоритм Киркпатрика-Зейделя

Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре
- 2 если $x_1 = x_2$, исключим точку с меньшей координатой y
- 3 для остальных пар посчитаем **наклоны**:

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

- 4 найдем медиану K множества наклонов k
- 5 проведем прямые с наклоном K через все точки, выберем верхнюю из них

Алгоритм Киркпатрика-Зейделя

Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре
- 2 если $x_1 = x_2$, исключим точку с меньшей координатой y
- 3 для остальных пар посчитаем **наклоны**:

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

- 4 найдем медиану K множества наклонов k
- 5 проведем прямые с наклоном K через все точки, выберем верхнюю из них
- 6 проверим, содержит ли эта прямая мост

Алгоритм Киркпатрика-Зейделя

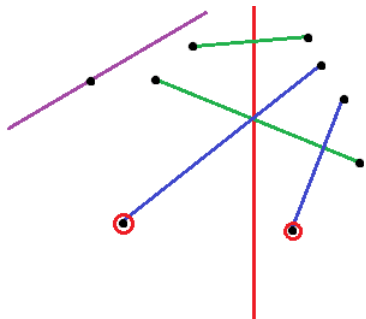
Как найти мост?

- 1 разобьем точки на пары, пусть $x_1 \leq x_2$ в каждой паре
- 2 если $x_1 = x_2$, исключим точку с меньшей координатой y
- 3 для остальных пар посчитаем **наклоны**:

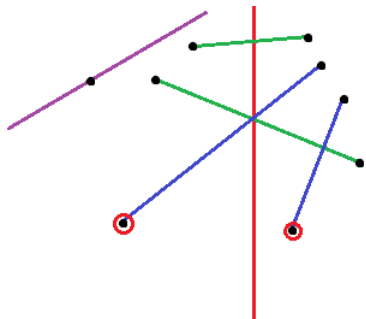
$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

- 4 найдем медиану K множества наклонов k
- 5 проведем прямые с наклоном K через все точки, выберем верхнюю из них
- 6 проверим, содержит ли эта прямая мост
- 7 иначе можно исключить из рассмотрения $\frac{1}{4}$ часть точек

Алгоритм Киркпатрика-Зейделя



Алгоритм Киркпатрика-Зейделя



$$f(n) = f\left(\frac{3n}{4}\right) + O(n)$$

Другой алгоритм за $O(n \log h)$.

Другой алгоритм за $O(n \log h)$.

1 предположим, размер выпуклой оболочки h известен

Другой алгоритм за $O(n \log h)$.

- 1 предположим, размер выпуклой оболочки h известен
- 2 разделим заданное множество точек на $n/h + 1$ частей размера не более h

Другой алгоритм за $O(n \log h)$.

- 1 предположим, размер выпуклой оболочки h известен
- 2 разделим заданное множество точек на $n/h + 1$ частей размера не более h
- 3 найдем выпуклую оболочку для каждой части при помощи алгоритма за $O(n \log n)$

Другой алгоритм за $O(n \log h)$.

- 1 предположим, размер выпуклой оболочки h известен
- 2 разделим заданное множество точек на $n/h + 1$ частей размера не более h
- 3 найдем выпуклую оболочку для каждой части при помощи алгоритма за $O(n \log n)$ (это займет $O(n/h)O(h \log h) = O(n \log h)$ времени)

Другой алгоритм за $O(n \log h)$.

- 1 предположим, размер выпуклой оболочки h известен
- 2 разделим заданное множество точек на $n/h + 1$ частей размера не более h
- 3 найдем выпуклую оболочку для каждой части при помощи алгоритма за $O(n \log n)$ (это займет $O(n/h)O(h \log h) = O(n \log h)$ времени)
- 4 используем алгоритм Джарвиса, чтобы объединить найденные выпуклые оболочки

Другой алгоритм за $O(n \log h)$.

- 1 предположим, размер выпуклой оболочки h известен
- 2 разделим заданное множество точек на $n/h + 1$ частей размера не более h
- 3 найдем выпуклую оболочку для каждой части при помощи алгоритма за $O(n \log n)$ (это займет $O(n/h)O(h \log h) = O(n \log h)$ времени)
- 4 используем алгоритм Джарвиса, чтобы объединить найденные выпуклые оболочки (h шагов $\times n/h$ многоугольников на каждом шаге $\times O(\log h)$ операций бинарного поиска $= O(n \log h)$)

Что делать, если мы не знаем h ?

Алгоритм Чана

Что делать, если мы не знаем h ?

Перебрать $h = 2^{2^t}$, $t = 0, 1, 2, \dots$

Что делать, если мы не знаем h ?

Перебрать $h = 2^{2^t}$, $t = 0, 1, 2, \dots$

Тогда общая вычислительная сложность

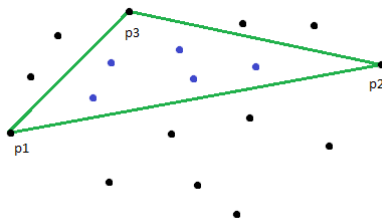
$$\sum_{t=0}^{\lceil \log \log h \rceil} O(n \log 2^{2^t}) = O(n) \sum_{t=0}^{\lceil \log \log h \rceil} O(2^t) = O(n \log h).$$

QuickHull

- 1 найдем самую левую и самую правую точки, они принадлежат выпуклой оболочке
- 2 соединим их базовой прямой и рекурсивно построим выпуклые оболочки двух множеств, разделенных этой прямой

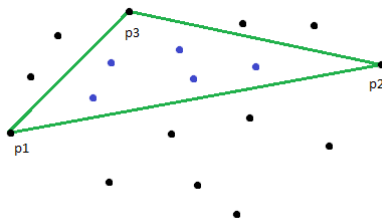
QuickHull

- 1 найдем самую левую и самую правую точки, они принадлежат выпуклой оболочке
- 2 соединим их базовой прямой и рекурсивно построим выпуклые оболочки двух множеств, разделенных этой прямой
- 3 найдем точку, максимально удаленную от базовой прямой; она также принадлежит выпуклой оболочке



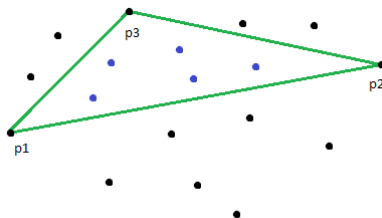
QuickHull

- 1 найдем самую левую и самую правую точки, они принадлежат выпуклой оболочке
- 2 соединим их базовой прямой и рекурсивно построим выпуклые оболочки двух множеств, разделенных этой прямой
- 3 найдем точку, максимально удаленную от базовой прямой; она также принадлежит выпуклой оболочке
- 4 удалим точки, лежащие внутри полученного треугольника



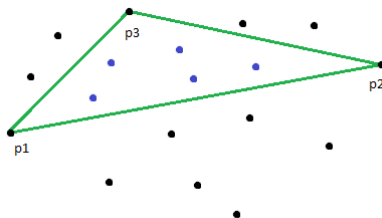
QuickHull

- 1 найдем самую левую и самую правую точки, они принадлежат выпуклой оболочке
- 2 соединим их базовой прямой и рекурсивно построим выпуклые оболочки двух множеств, разделенных этой прямой
- 3 найдем точку, максимально удаленную от базовой прямой; она также принадлежит выпуклой оболочке
- 4 удалим точки, лежащие внутри полученного треугольника
- 5 решим задачу для двух новых сторон треугольника рекурсивно



QuickHull

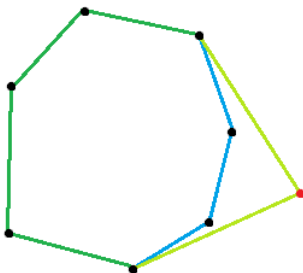
- 1 найдем самую левую и самую правую точки, они принадлежат выпуклой оболочке
- 2 соединим их базовой прямой и рекурсивно построим выпуклые оболочки двух множеств, разделенных этой прямой
- 3 найдем точку, максимально удаленную от базовой прямой; она также принадлежит выпуклой оболочке
- 4 удалим точки, лежащие внутри полученного треугольника
- 5 решим задачу для двух новых сторон треугольника рекурсивно



Вычислительная сложность $O(n \log n)$ в среднем, $O(n^2)$ в худшем случае.

Добавление точек по одной

- 1 начнем с трех точек, образующих треугольник
- 2 будем добавлять точки по одной
- 3 если точка находится снаружи выпуклой оболочки, удалим все стороны, которые **видит** новая точка, и соединим точку с оставшейся частью выпуклой оболочки



Вычислительная сложность $O(n^2)$.

Выпуклая оболочка 3D

Общие факты

- выпуклая оболочка в 3D — это многогранник

Общие факты

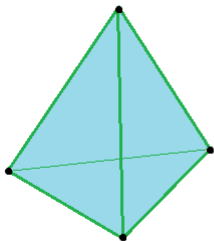
- выпуклая оболочка в 3D — это многогранник
- его грани можно триангулировать, поэтому мы будем работать с множеством треугольников

Общие факты

- выпуклая оболочка в 3D — это многогранник
- его грани можно триангулировать, поэтому мы будем работать с множеством треугольников
- количество ребер и граней составляет $O(\text{Вершин})$, это следствие из **формулы Эйлера**: $\text{Вершины} - \text{Ребра} + \text{Грани} = 2$

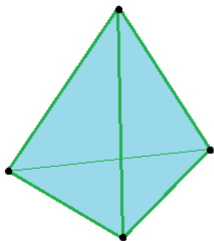
Добавление точек по одной

1 начнем с тетраэдра



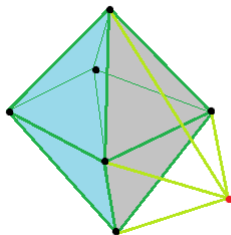
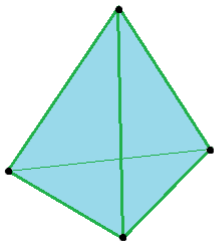
Добавление точек по одной

- 1 начнем с тетраэдра
- 2 будем добавлять точки по одной



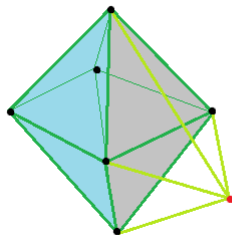
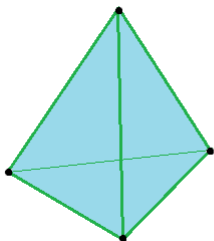
Добавление точек по одной

- 1 начнем с тетраэдра
- 2 будем добавлять точки по одной
- 3 определим треугольные грани, которые **видит** новая точка



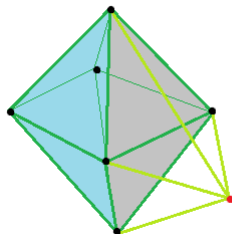
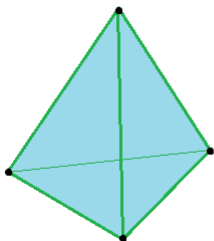
Добавление точек по одной

- 1 начнем с тетраэдра
- 2 будем добавлять точки по одной
- 3 определим треугольные грани, которые **видит** новая точка (будем хранить для каждой грани уравнение плоскости $Ax + By + Cz + D$ с определенной ориентацией)



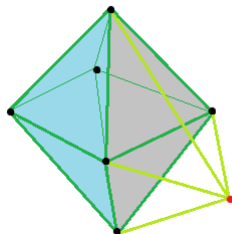
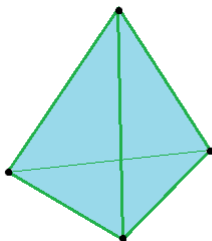
Добавление точек по одной

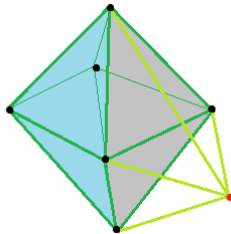
- 1 начнем с тетраэдра
- 2 будем добавлять точки по одной
- 3 определим треугольные грани, которые **видит** новая точка (будем хранить для каждой грани уравнение плоскости $Ax + By + Cz + D$ с определенной ориентацией)
- 4 если таких граней не существует, то точка внутри выпуклой оболочки



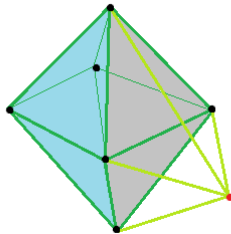
Добавление точек по одной

- 1 начнем с тетраэдра
- 2 будем добавлять точки по одной
- 3 определим треугольные грани, которые **видит** новая точка (будем хранить для каждой грани уравнение плоскости $Ax + By + Cz + D$ с определенной ориентацией)
- 4 если таких граней не существует, то точка внутри выпуклой оболочки
- 5 иначе нужно удалить некоторые грани и добавить некоторые новые



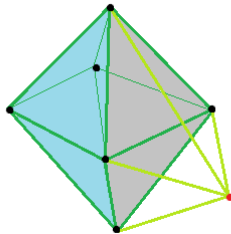


6 при удалении грани пометим три соответствующих ребра

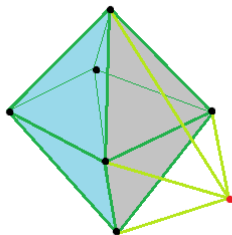


6 при удалении грани пометим три соответствующих ребра

7 если ребро помечено дважды, удаляем его



- 6 при удалении грани пометим три соответствующих ребра
- 7 если ребро помечено дважды, удаляем его
- 8 если ребро помечено только один раз, добавим новую треугольную грань, образованную этим ребром и новой точкой



- 6 при удалении грани пометим три соответствующих ребра
- 7 если ребро помечено дважды, удаляем его
- 8 если ребро помечено только один раз, добавим новую треугольную грань, образованную этим ребром и новой точкой

Вычислительная сложность $O(n^2)$.

- 1 выберем четыре крайние точки и построим тетраэдр

QuickHull

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра

QuickHull

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани

QuickHull

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани
- 4 будем обрабатывать грани по одной, у каждой грани есть свой список точек

QuickHull

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани
- 4 будем обрабатывать грани по одной, у каждой грани есть свой список точек
- 5 найдем наиболее удаленную точку в списке грани

QuickHull

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани
- 4 будем обрабатывать грани по одной, у каждой грани есть свой список точек
- 5 найдем наиболее удаленную точку в списке грани
- 6 эта точка может видеть другие грани! удалим все такие грани

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани
- 4 будем обрабатывать грани по одной, у каждой грани есть свой список точек
- 5 найдем наиболее удаленную точку в списке грани
- 6 эта точка может видеть другие грани! удалим все такие грани
- 7 пометим ребра удаленных граней
- 8 если ребро помечено дважды, удалим его
- 9 если ребро помечено только один раз, создадим новую грань, образованную этим ребром и новой точкой

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани
- 4 будем обрабатывать грани по одной, у каждой грани есть свой список точек
- 5 найдем наиболее удаленную точку в списке грани
- 6 эта точка может видеть другие грани! удалим все такие грани
- 7 пометим ребра удаленных граней
- 8 если ребро помечено дважды, удалим его
- 9 если ребро помечено только один раз, создадим новую грань, образованную этим ребром и новой точкой
- 10 точки из списков удаленных граней должны быть перераспределены между новыми гранями или удалены

QuickHull

- 1 выберем четыре крайние точки и построим тетраэдр
- 2 удалим точки, оказавшиеся внутри тетраэдра
- 3 остальные точки распределим между треугольными гранями, которые они **могут видеть**; каждую точку отдадим в точности одной грани
- 4 будем обрабатывать грани по одной, у каждой грани есть свой список точек
- 5 найдем наиболее удаленную точку в списке грани
- 6 эта точка может видеть другие грани! удалим все такие грани
- 7 пометим ребра удаленных граней
- 8 если ребро помечено дважды, удалим его
- 9 если ребро помечено только один раз, создадим новую грань, образованную этим ребром и новой точкой
- 10 точки из списков удаленных граней должны быть перераспределены между новыми гранями или удалены

Вычислительная сложность $O(n^2)$ в худшем случае, $O(n \log n)$ в среднем.

Алгоритм Джарвиса

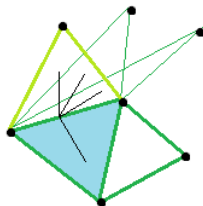
- 1 найдем стартовую грань

Алгоритм Джарвиса

- 1 найдем стартовую грань
- 2 обработаем ее ребра

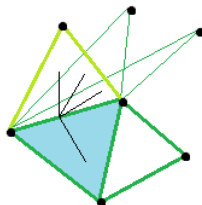
Алгоритм Джарвиса

- 1 найдем стартовую грань
- 2 обработаем ее ребра
- 3 ребро образует плоскость с каждой из оставшихся точек



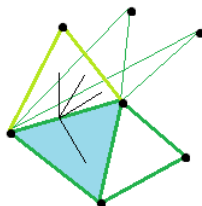
Алгоритм Джарвиса

- 1 найдем стартовую грань
- 2 обработаем ее ребра
- 3 ребро образует плоскость с каждой из оставшихся точек
- 4 выберем точку, для которой угол между этой плоскостью и рассматриваемой в данный момент гранью выпуклой оболочки максимален



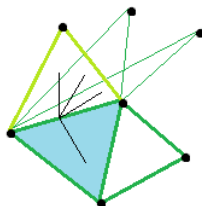
Алгоритм Джарвиса

- 1 найдем стартовую грань
- 2 обработаем ее ребра
- 3 ребро образует плоскость с каждой из оставшихся точек
- 4 выберем точку, для которой угол между этой плоскостью и рассматриваемой в данный момент гранью выпуклой оболочки максимален
- 5 добавим новую грань и продолжим обрабатывать грани в порядке обхода в глубину или обхода в ширину



Алгоритм Джарвиса

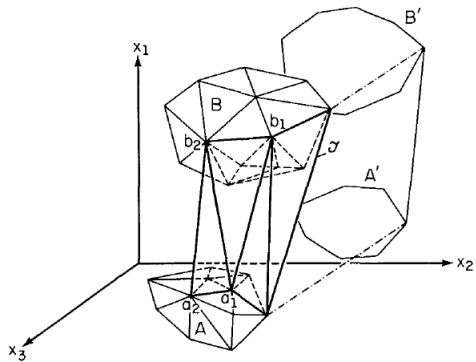
- 1 найдем стартовую грань
- 2 обработаем ее ребра
- 3 ребро образует плоскость с каждой из оставшихся точек
- 4 выберем точку, для которой угол между этой плоскостью и рассматриваемой в данный момент гранью выпуклой оболочки максимален
- 5 добавим новую грань и продолжим обрабатывать грани в порядке обхода в глубину или обхода в ширину



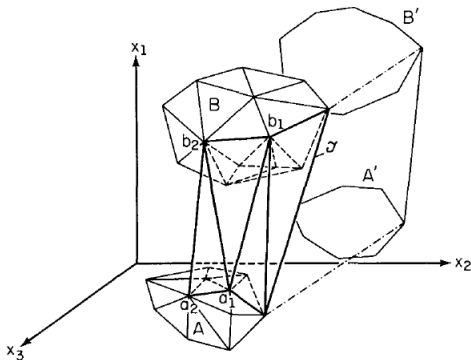
Вычислительная сложность алгоритма $O(nh)$.

«Разделяй и властвуй» (алгоритм Препараты)

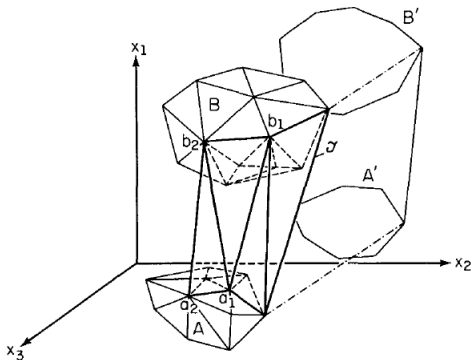
Работает по тому же принципу, что и «разделяй и властвуй» в 2D.
Основной вопрос состоит в том, **как объединить два многогранника за время $O(n)$?**



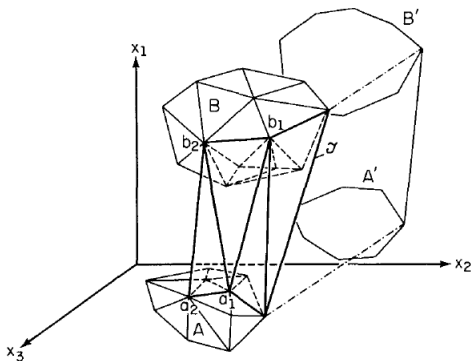
- 1 построим проекции многогранников на плоскость и найдем стартовый отрезок «цилиндра»



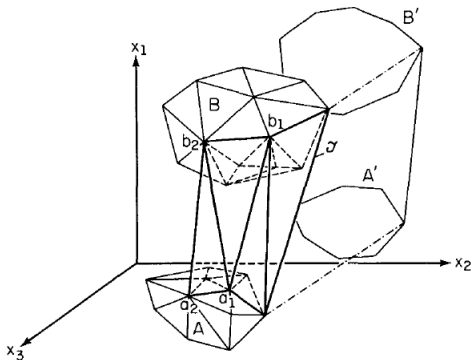
- 1 построим проекции многогранников на плоскость и найдем стартовый отрезок «цилиндра»
- 2 переберем треугольники, образованные текущим отрезком и «кандидатами» из числа точек обоих многогранников



- 1 построим проекции многогранников на плоскость и найдем стартовый отрезок «цилиндра»
- 2 переберем треугольники, образованные текущим отрезком и «кандидатами» из числа точек обоих многогранников
- 3 каждый «кандидат» должен быть соединен ребром с одним из концов текущего отрезка



- 1 построим проекции многогранников на плоскость и найдем стартовый отрезок «цилиндра»
- 2 переберем треугольники, образованные текущим отрезком и «кандидатами» из числа точек обоих многогранников
- 3 каждый «кандидат» должен быть соединен ребром с одним из концов текущего отрезка
- 4 выберем плоскость, образующую наибольший угол с последней построенной гранью «цилиндра»



«Разделяй и властвуй» (алгоритм Препараты)

Preparata F.P., Hong J.S. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. Communications of the ACM, Volume 20, Issue 2, Feb. 1977, Pages 87-93.

Вычислительная сложность алгоритма $O(n \log n)$.

«Разделяй и властвуй» (алгоритм Препараты)

Preparata F.P., Hong J.S. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. Communications of the ACM, Volume 20, Issue 2, Feb. 1977, Pages 87-93.

Вычислительная сложность алгоритма $O(n \log n)$.

Используя этот алгоритм, Чан предложил алгоритм за $O(n \log h)$ в 3D.

Chan T.M. Optimal Output-Sensitive Convex Hull Algorithms in Two and Three dimensions. Discrete Comput. Geom. 16: 361-368 (1996).

Спасибо!