QuickHull

Introducción:

QuickHull es un algoritmo que posee un enfoque divide and conquer y recibe su nombre por su similitud con el algoritmo de ordenamiento QuickSort. El algoritmo sigue la intuición que dice que es más sencillo crear el casco convexo de forma incremental, si eliminamos los vértices que tenemos asegurado que serán internos al casco convexo final y nos concentramos en agregar los vértices cercanos a las fronteras. El algoritmo comienza su ejecución creado el primer casco convexo temporal $\boldsymbol{C_0}$, el cual esta formado por dos maximales del conjunto de puntos, por ejemplo el maximal izquierdo y el maximal derecho. Podemos ver este primer paso en la Ilustración 1.

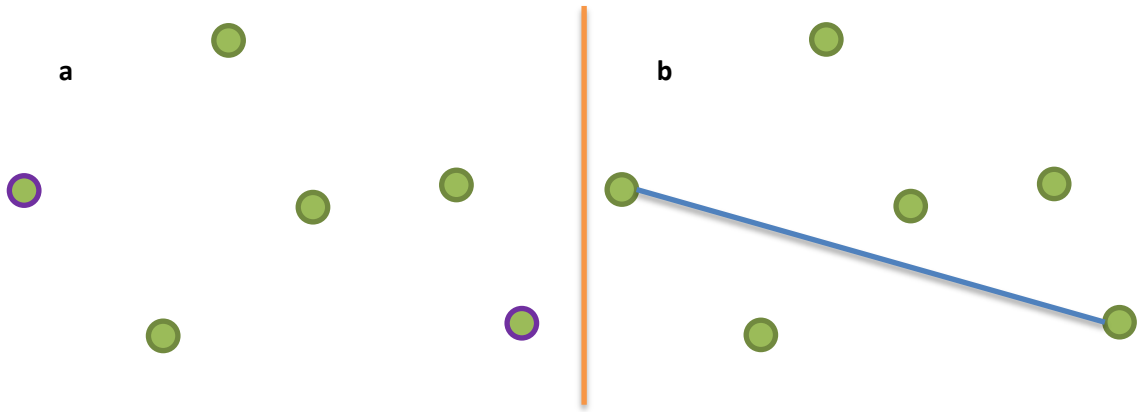

Ilustración 1: a) Elección (morado) de maximales b) Creación del casco convexo temporal C_0

El próximo paso es usar las normales del convex hull temporal $\boldsymbol{C_0}$ para buscar los vértices más alejados al plano que forma la arista del casco. Esos puntos son agregados al convex hull temporal que será llamado $\boldsymbol{C_1}$ al extender con una normal y luego $\boldsymbol{C_2}$ al extender por la otra normal. Podemos ver eso en la Ilustración 2.

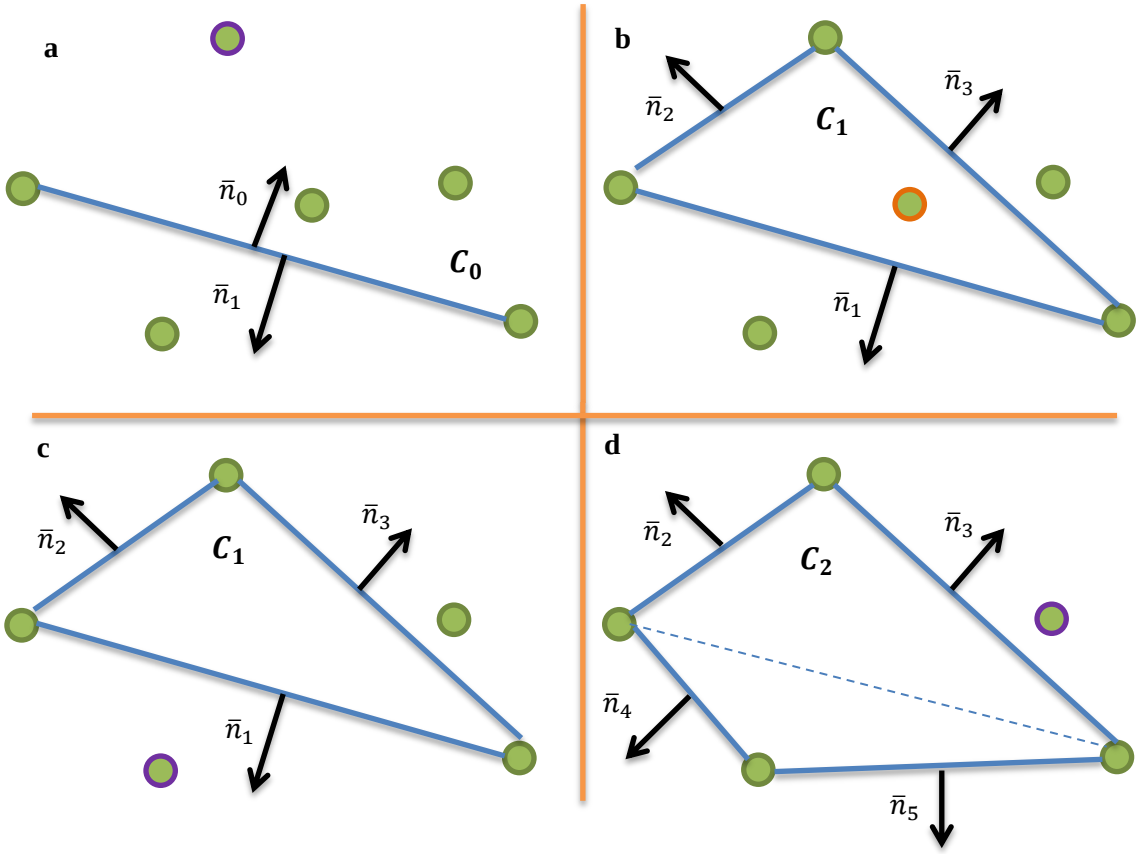

Ilustración 2: a) Marcado de normales del casco y vértice mas lejano (morado) b) Agrega vertice mas lejano al casco y elimina vetices internos al casco (naranja) c) Busca vertice mas lejano a normal (morado) d) Agrega vertice a casco convexo

El algoritmo sigue hasta que no queda ningún vértice para agregar en la cola de normales. El pseudocódigo es:

I. ***Buscar 2 maximales distintos, formar el casco convexo inicial con estos y poner sus dos normales en la cola de normales CN.***

II. ***Por cada normal n de la cola de normales CN***

- ***a. Buscar el vértice mas alejado(positivamente) en la dirección de esa normal en el conjunto V; si este no existe luego continua procesando la próxima normal de CN, si existe este vértice será llamado*** V_{far}
- ***b. Ahora formar un triangulo con los vértices de la normal*** V_a ***y*** V_b***, y*** V_{far}***; luego eliminar de la lista de vértices V los vértices internos al triangulo***
- ***c. Agregar a CN las dos normales nuevas que se producirán por agregar*** V_{far} ***al casco convexo***

Análisis de correctitud:

Primero debemos notar que al unir dos maximales diferentes al convex hull inicial, estamos agregando dos vértices que sabemos que pertenecerán al convex hull final. Luego probaremos que *cada vez que se agrega un vértice el casco convexo temporal mantiene la convexidad* y que *solo se descartan vértices internos al convex hull final*, finalmente probaremos que *se agregan todos los vértices del convex hull*.

- ***Cuando se agregan vértices a C_i mantiene la convexidad?***

Al agregar un vértice, como dijimos es el mas alejado en un determinada dirección que es determinada por la normal de la arista. Por lo tanto una línea que define un semiplano tal que todos los puntos $\boldsymbol{V}$ están contenidos en éste. Podemos ver en la Ilustración 3 dos posibles rectas que definen el semiplano para el vértice $\boldsymbol{V_a}$. Algo análogo sucede con $\boldsymbol{V_b}$. Además sabemos que para que al agregar un vértice al casco $\boldsymbol{C_i}$, éste siga siendo convexo el vértoce a agregar debe ser interior a las tangentes que definen las aristas vecinas a la normal en cuestión. Podemos ver esas tangentes en Ilustración 3 como líneas interrumpidas. Lo importante es que debido a las rectas, verdes, de las que hablamos anteriormente no puede haber vértices fuera de esta área que definen las tangentes a las aristas, líneas interrumpidas. Por lo tanto todo vértice que se agregue al convex hull temporal mantendrá la convexidad del casco.

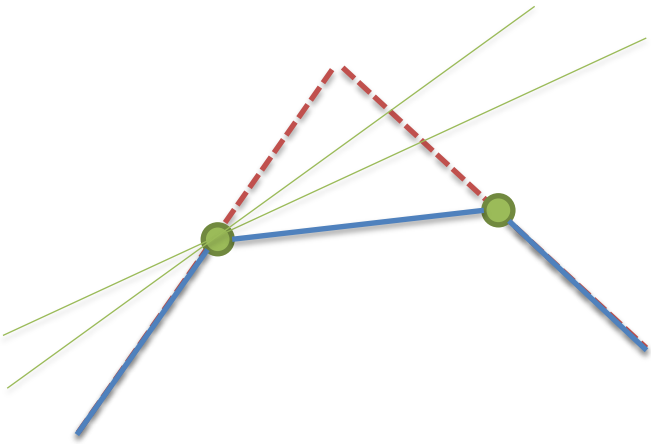

Ilustración 3: Posibles posiciones de vértices en fronteras a expandir

- ***Solo se eliminan vértices internos?***

Los vértices que se eliminan sabemos que eran internos al casco $\mathbf{C_{i+1}}$, ya que ese era el criterio de descarte que usamos. También sabemos que $\mathbf{C_{i+1}}$, el cual acabamos de demostrar que era convexo, así que solo se están eliminando vértices internos a un subcasco convexo del final, es decir un subconjunto del casco convexo final.

- ***Se agregan todos los puntos del convex hull?***

Si un punto, llamémoslo $\overline{V_x}$, que pertenecía al convex hull final no se agregó, eso significa que no estaba a una distancia positiva de alguna de las normales que forman nuestro casco C_{final}, ya que el algoritmo busca en todas las normales finales de ese casco. Lo anterior implica que el vértice $\overline{V_x}$ debe estar a una distancia negativa de todas las normales (planos) del casco C_{final}, pero si sucede esto eso quiere decir que $\overline{V_x}$ estaba dentro del casco C_{final}, i.e. $\overline{V_x}$ no puede existir.

Casos patológicos:

a. Vértices encimados

Si pudiesen existir dos vértices encimados en el casco convexo final, sean estos $\overline{V_a}$ y $\overline{V_b}$, eso significaría que los dos vértices fueron identificados como los más alejados a una normal en el proceso de construcción del casco convexo. En el proceso de construcción eso no pasara ya que cualquier normal identificara o a uno mas alejado por un motivo numérico o como a igual distancia, lo cual hará que elija a cualquiera de los dos, pero a solo uno. La única normal que puede agregar el otro vértice luego es alguna de las normales que se forma con el vértice que fue recién agregado, y el motivo de agregarlo seria un pequeño error numérico que diga que el otro vértice esta a una distancia muy pequeña de la normal, cuando en realidad esta a distancia cero. La solución a esto es solo agregar vértices más alejados a una normal si es que estos están a una distancia mayor a un ε prefijado. Para solucionar el problema de que se considere como iguales a vértices que están verdaderamente a una distancia menor a ε entre si, se puede agregar una perturbación simbólica a estos vértices para alejarlos un ε, lo cual hará que no se los considere iguales luego. Todavía esta el problema de que esas perturbaciones simbólicas una vez aplicadas hagan que 2 o mas vértices simbólicamente perturbados que estaban separados por ε puedan entra en el rango de ser considerados iguales, este es un problema con el cual tendremos que vivir. Una solución seria aplicar iterativamente perturbaciones hasta que no se de el problema explicado anteriormente, pero aplicar iterativamente perturbaciones simbólicas de forma naive no garantiza la convergencia y por lo tanto puede que entre en un ciclo infinito.

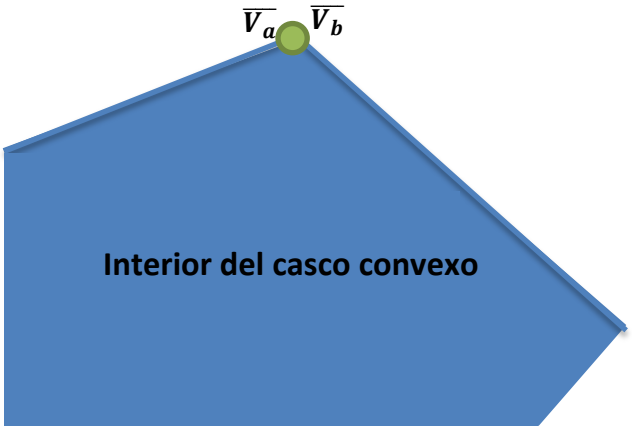

Ilustración 4: Vértices encimados

b. Vértices alineados en la frontera

A este problema para no agregar el vértice intermedio $\overline{V_b}$, se puede aplicar la misma solución que el problema de los vértices encimados, ya que $\overline{V_b}$ deberá ser identificado como el más alejado a la normal ya encontrada, de igual forma que el vértice encimado. La diferencia radica en que en vez de aplicar la perturbación simbólica al comienzo del algoritmo, ahora se la aplicara ante la búsqueda del vértice mas alejado de una normal, si esta alineado, i.e. en el plano, no se le hace nada y no se agrega por no estar a un $\boldsymbol{\varepsilon}$ de la normal, pero si no estaba alineado se le aplica un $\boldsymbol{\varepsilon}$ alejándolo.

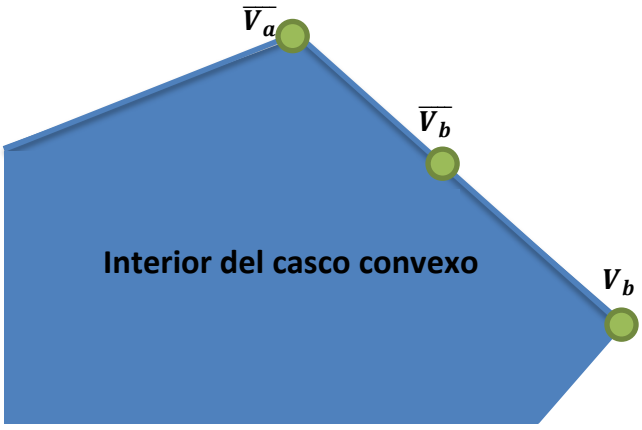

Ilustración 5: Vértices alineados en la frontera

c. Vértices equidistantes a la arista

Cuando se busca el punto más alejado de una arista respecto a su normal, puede haber más de un punto a la misma distancia, de forma tal que cualquiera puede tomarse como el vértice

a agregar. Esto es un problema si no se quiere que el casco convexo contenga puntos alineados. Una posible solución es, de estos puntos equidistantes, elegir el máximo o mínimo respecto a la dirección perpendicular a la normal que se está utilizando (como un orden lexicográfico rotado), de manera que nunca agreguemos un punto del medio, ya que esto último es lo que podría causar puntos alineados en el casco convexo final.

Análisis de tiempo:

En cada iteración del algoritmo se realizan $\boldsymbol{\theta(n)}$ operaciones. Además, en cada iteración se agrega un vértice del casco convexo o bien se llegó a una arista del casco. En todo caso, la cantidad de iteraciones depende linealmente de la cantidad de vértices del casco convexo. Esta cantidad es de orden $\boldsymbol{\theta(\log n)}$ para el caso promedio, por lo que el tiempo total del algoritmo es de $\boldsymbol{\theta(n \log n)}$. El peor caso sucede cuando todos los puntos dados pertenecen al casco convexo final, en cuyo caso el orden es de $\boldsymbol{\theta(n^2)}$.