How to use Fast QHull

Andrea Casalino

December 3, 2019

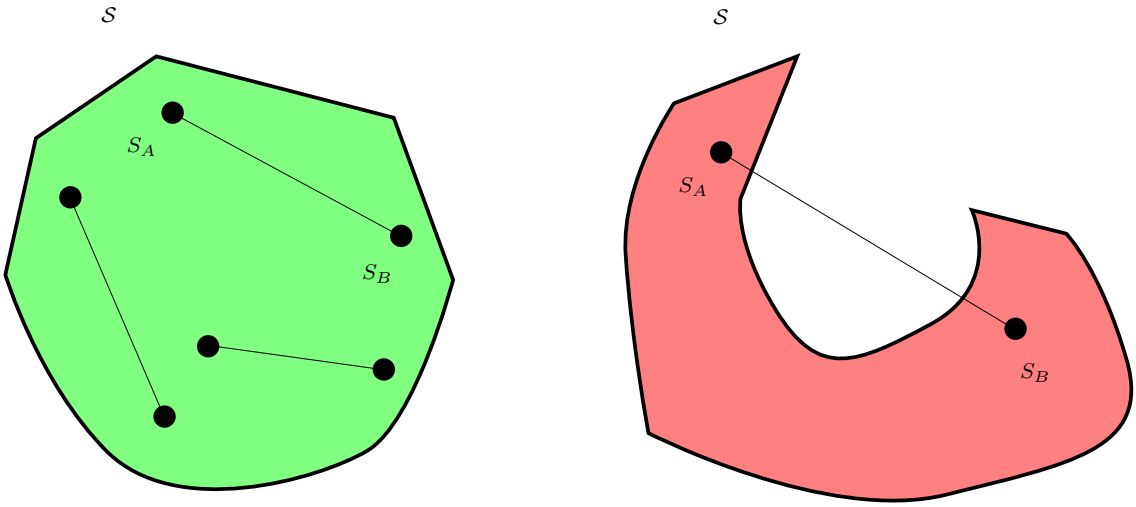


Figure 1: Examples of a convex shape (left) and a non convex one (right). The segment connecting any pair of points within the shape, it's entirely contained in the shape itself. This is not true for the non convex example on the right.

1 What is a convex hull?

The Convex hull of a shape [1] is the minimum convex volume entirely containing that shape. At this point it is important to define the concept of convexity. The generic convex shape $\mathcal{S} = \{S_1, \cdots, S_N\}$, is a set of points for which it holds that:

$$\mathcal{S}\ is\ convex \quad \Rightarrow \quad \forall S_{A,B} \in \mathcal{S} \wedge \forall r \in [0,1] \tag{1}$$

$$\Rightarrow \quad S_A + r \cdot (S_B - S_A) = S(r) \in \mathcal{S} \tag{2}$$

i.e. the segment connecting the generic pair of points $S_A, S_B \in \mathcal{S}$, is entirely contained in $\mathcal{S}$. Examples of convex and non convex shapes are reported in Figure 1. Clearly, the convex hull of a convex object it's the object itself. On the opposite, when dealing with any other kind of shapes, the computation of the convex hull is non trivial to perform. Examples of convex hulls wrapping non convex shapes are reported in Figure 2.

2 Convex hull representation: meshes

In case of 3 dimensional objects, the convex hull is represented by a set of triangular facets, forming a convex mesh. Each facet in the mesh is characterized by the positions of the three vertices delimiting that facet. Since the vertices are shared by the facets, an efficient representation of a mesh consists of a bipartite set $\langle \mathcal{V}, \mathcal{I} \rangle$. $\mathcal{V} = \{V_1, \cdots, V_N\}$ defines the coordinates of the vertices pertaining to the mesh, while $\mathcal{I}$ are the so called incidences. The j^{th} element $I_j \in \mathcal{I}$, is a tuple $I_j = \langle V_a^j, V_b^j, V_c^j \rangle$ specifying the vertices delimiting the j^{th} facet. Figure 3 reports example of mesh representation. Notice that the above representation is able to describe both convex and non-convex meshes. However, in case of convex hull, the resulting mesh will be always convex.

3 Convex hull determination

FastQHull was conceived to compute the convex hull of a shape whose geometry is described by the coordinates of a point cloud $\mathcal{C} = \{C_1, \cdots, C_L\}$. If you are interested in determining the convex hull of a solid object, described for example by a (non convex) mesh, pass as $\mathcal{C}$ the set of vertices pertaining to that solid. Clearly, the resulting convex hull $\{\mathcal{V}_{CH}, \mathcal{I}_{CH}\}$ will be such that $\mathcal{V}_{CH} \subseteq \mathcal{C}$ or in other words, the vertices of the convex hull are a subset of the initial point cloud. The convex hull computation is subdivided into two main steps, described in Sections 3.1 and 3.2.

[1] The term shape will refer in this document to a generic collection of points in the space. It can be either a compact or a non compact set like for example a set of disconnected vertices.

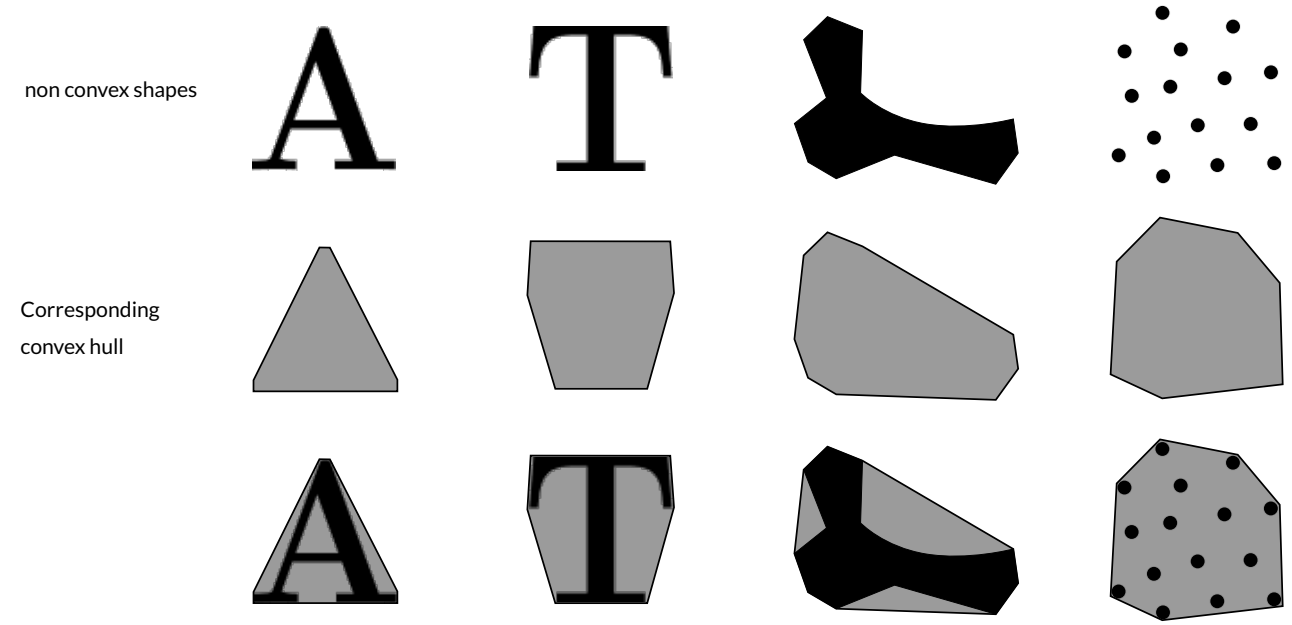
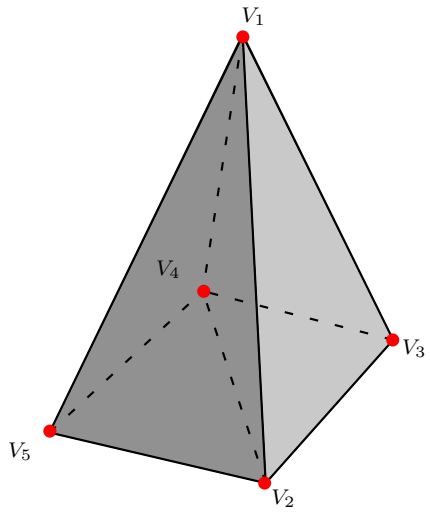
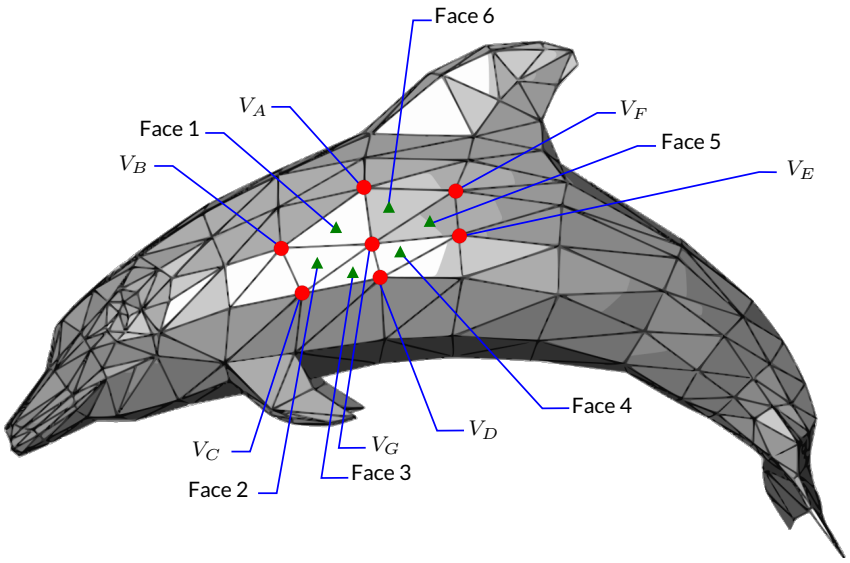


Figure 2: Examples of convex hulls.



$$\mathcal{I} = \{ \quad \langle V_1, V_2, V_3 \rangle \\ \langle V_1, V_3, V_4 \rangle \\ \langle V_1, V_5, V_4 \rangle \\ \langle V_1, V_2, V_5 \rangle \\ \langle V_2, V_3, V_4 \rangle \\ \langle V_2, V_4, V_5 \rangle \quad \} \tag{3}$$



$$\mathcal{I} = \{ \quad \langle V_A, V_B, V_G \rangle \\ \langle V_B, V_C, V_G \rangle \\ \langle V_C, V_D, V_G \rangle \\ \langle V_D, V_E, V_G \rangle \\ \langle V_E, V_F, V_G \rangle \\ \langle V_F, V_A, V_G \rangle \quad \} \tag{4}$$

Figure 3: Examples of mesh representation. On the left a simple convex object and it's incidences, while on the right a non convex shape with the incidences of some of their facets..

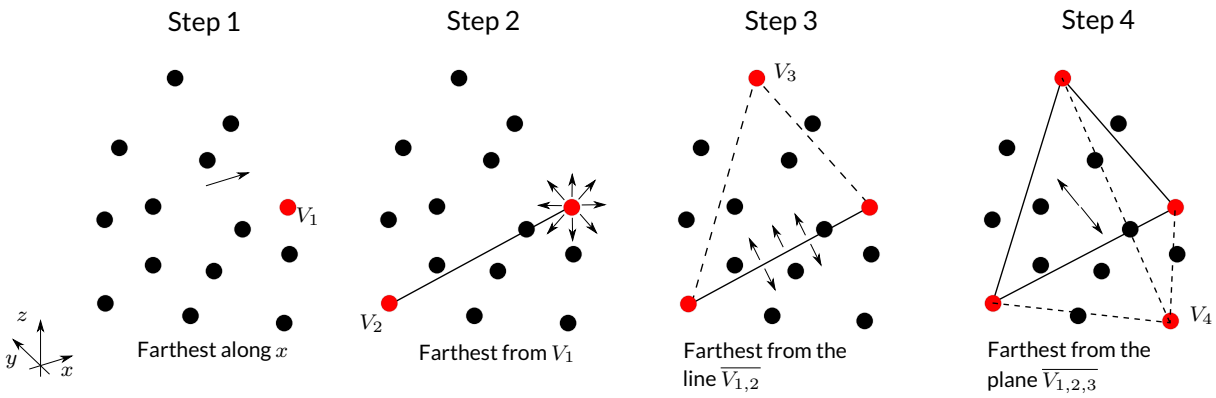


Figure 4: Steps involved in the determination of the initial tetrahedron.

3.1 Phase 1

The algorithm starts by defining an initial tetrahedron that will be then iteratively expanded till converging to the convex hull. The 4 vertices $\{V_1, V_2, V_3, V_4\}$ of the initial tetrahedron are found as described in the following.
V_1 is assumed equal to [2]

$$V_1 = argmax_{C_i}\{< \hat{x}, C_i >\} \tag{5}$$

with $\hat{x}$ that is the x axis versor, i.e. $\hat{x} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}$. Essentially, V_1 is identified as the point in the cloud having the maximum value for the x coordinate. V_2 is computed as:

$$V_2 = argmax_{C_i}\{< C_i - V_1, C_i - V_1 >\} \tag{6}$$

i.e. the point in the cloud maximising the Euclidean distance w.r.t. V_1. V_3 is then computed as:

$$V_3 = argmax_{C_i}\{dist(C_i, \overline{V_{12}})\} \tag{7}$$

where $dist(C_i, \overline{V_{12}})$ stands for the distance between C_i and the line passing between V_1 and V_2 (see `https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line`). Finally, V_4 is assumed as:

$$V_4 = argmax_{C_i}\{dist(C_i, \overline{V_{123}})\} \tag{8}$$

where $dist(C_i, \overline{V_{12}})$ stands for the distance between C_i and the plane identified by V_1,V_2 and V_3 (see `https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_plane`). Figure 4 summarizes the above steps. The volume A of the tetrahedron can be computed by considering a triple product (`https://en.wikipedia.org/wiki/Triple_product`) [3]

$$A =< V_3 - V_4, (V_2 - V_4) \wedge (V_1 - V_4) > \tag{9}$$

In case A results to be close to 0 (or at least 0), it means that the cloud $\mathcal{C}$ is entirely contained in a single plane or a line. In such degenerate cases, the convex hull computations cannot be performed and the procedure is interrupted.

3.2 Phase 2

The initial tetrahedron is iteratively updated. At the generic step k, the intermediate result $\{\mathcal{V}_{CH}, \mathcal{I}_{CH}\}_k$ must be evaluated. The distance D_j of the farthest point from the j^{th} facet in $\{\mathcal{V}_{CH}, \mathcal{I}_{CH}\}_k$ is defined as follows:

$$D_j = max_{C_i}\{d(C_i, j)\} \tag{10}$$

$$d(C_i, j) = min(0, \langle C_i - V_a^j, n^j \rangle) \tag{11}$$

where n^j is the outer normal versor of the j^{th} facet, refer also to Figure 5. The facet having the maximum D^* is searched in the mesh $\{\mathcal{V}_{CH}, \mathcal{I}_{CH}\}_k$ and the corresponding farthest point $C^* \in \mathcal{C}$ is considered. A facet j is said to be visible from C^* when it holds that $d(C^*, j) > 0$. The update of the mesh consists in deleting the set of facets visible from the farthest point C^*, with the aim of replacing them with a cone of new facets. The process is described in Figure 6.
The algorithm stops when reaching a situation for which D^* is equal to 0. Indeed, in such a case all the points in the cloud are already wrapped and the convex hull is computed.

[2] $<,>$ stands for the dot product: $< a, b >= a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z$
[3] The operator $\wedge$ indicates the cross product of the two vectors: $c = a \wedge b = \begin{bmatrix} c_x = a_y \cdot b_z - a_z \cdot b_y & c_y = a_z \cdot b_x - a_x \cdot b_z & c_z = a_x \cdot b_y - a_y \cdot b_x \end{bmatrix}$

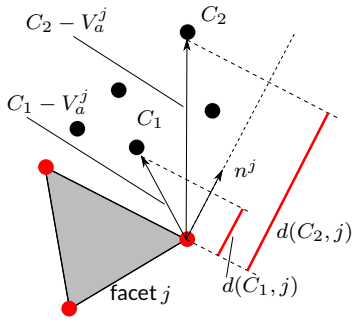


Figure 5: Computation of d.

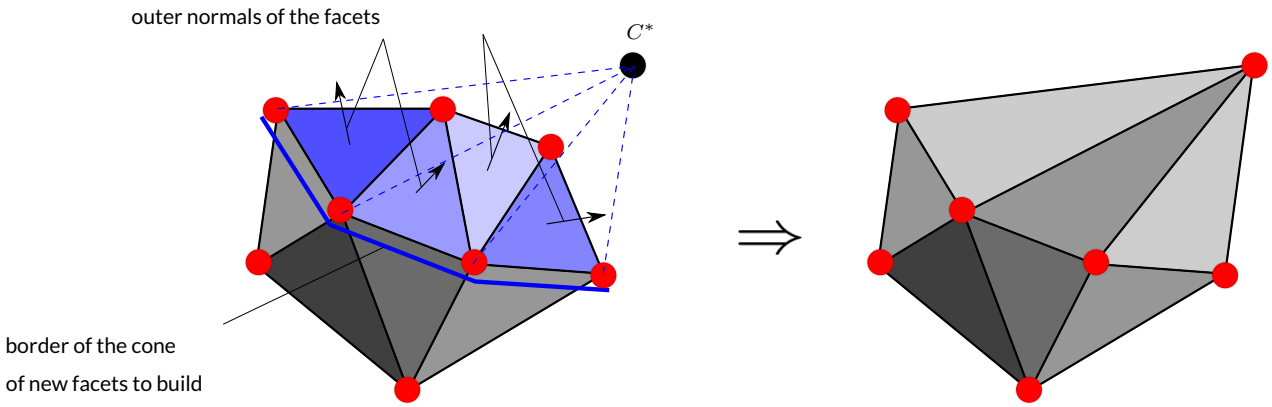


Figure 6: Convex hull update from step k to $k+1$. The set of visible facets (the blue ones on the left) is replaced with a cone of new facets (visible on the right).

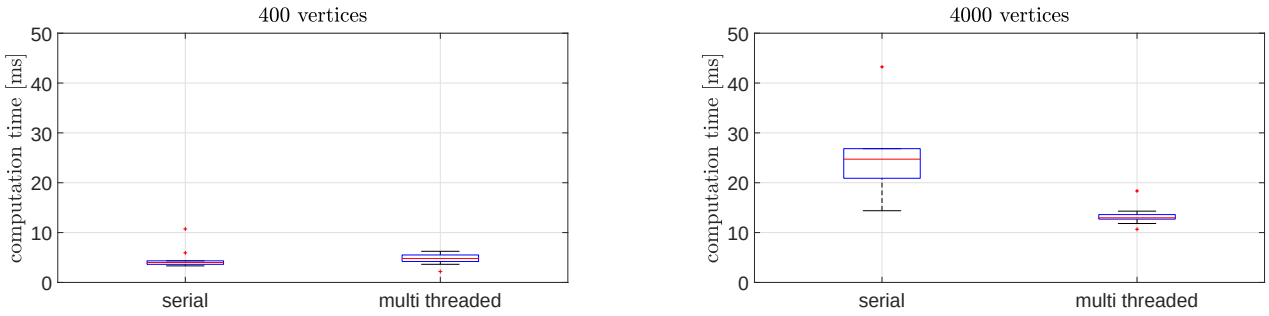


Figure 7: Statistics of the time required for computing the convex hull: in the left considering 20 sampled clouds made of 400 vertices, while on the right considering considering to sample clouds made of 4000 vertices. The multi threaded strategy seems to have significant better performance when having many points in the cloud.

4 Accelerating the algorithm using threads

The computation of the farthest point of a facet (Figure 5) can be significantly time consuming when considering clouds $\mathcal{C}$ made of thousands of points. In such cases, the farthest point search can be speed up by considering a multi threading strategy. Indeed, a parallel for (`http://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-loop.html`) can be employed to allow threads search in parallel the farthest point. It would be inefficient to rebuilt and destroy at every iteration of the convex hull update the threads. On the opposite, a thread pooling strategy (`https://en.wikipedia.org/wiki/Thread_pool`) was adopted. There is a single thread performing all the steps of the algorithm and many silent ones. When a new facet is built, the silent threads are woke up for performing, together with the main thread, the parallel search of the farthest point. The parallel region are handled adopting OpenMP (`https://www.openmp.org`). In particular, when OpenMP has been enabled from your compiler (as an example follow this link if you are using Visual Studio: `https://docs.microsoft.com/en-us/cpp/build/reference/openmp-enable-openmp-2-0-support?view=vs-2019`), FastQHull.h is automatically built with the parallel computing strategy described here. On the opposite, when OpenMP is not enabled, FastQHull.h is compiled with the serial standard version of the convex hull algorithm, adopting a single thread for searching the farthest points.
The parallel strategy is particularly convenient when dealing with clouds made of many vertices, while performance are comparable to the serial standard version when dealing with small-medium size clouds, refer to Figure 7.