Tsinghua
University

Computational Geometry

Construction of 3-Dimensional Convex Hull

jean-baptiste Debard [2004280017] <*yangf@cg.cs.tsinghua.edu.cn*>
gwenael Allard [2004280019] <*gaol@cg.cs.tsinghua.edu.cn*>
Lu-ying jie [2004310588] <*lu-yj04@mails.tsinghua.edu.cn*>
Compiled the 2 janvier 2005.

1 Description of the Algorithm

1.1 Specifications of the Project

1.1.1 Declaration

The project `ConvexHull` in the submit directory is developed by our team.Of the code concerning the 3D and 2D convex hull construction, which is the kernel of this project, we leave them totally free. But we would like to reserve the copyright to all other code in this project.

1.1.2 Test Data

Test data are provided in the `Example` directory

- `2DexampleXY.txt`, two dimensional example in xy plane
- `2DexampleYZ.txt`, two dimensional example in yz plane
- `3D_cube.txt`, a simple 3D example
- `3D_cube_plan.txt`, a simple 3D example where the first 4 node lie in a plane
- `pt_worst_case01.txt`, a worst case of constructing a 3D convex hull
- `pt_worst_case_random.txt`, the above case with its points randomly disordered.

1.1.3 User Manual

After start up, the user can rearrange the sub windows to his own preference by drag-and-drop. The user can then access the command panel for the operations, which are listed below.

- `Open Point File Open a data file`, user can save his data to disk, and can also generate the data using any text editor. In either case the file can be loaded here in this command.
- `Save Point File`, save the points to a file.
- `New Demo`, this is just a reset of the workspace
- `Random Points`, this is a random point generator, a dialog would appear and guide the user to generate some number of random points
- `CH Stepping`, construct the convex hull in a step by step manner.
- `Automatic Stepping`, run the steps automatically
- `Convex Hull`, Construct the convex hull in a single step
- `Show Input Points`, toggle showing of input points
- `Show Point Number`, toggle showing of the point number
- `Show Facets`, toggle showing of the facets
- `Show Edges`, toggle showing of the edge
- `Show Current Vertex`, toggle showing of the currently processing vertex
- `Show the Horizon`, toggle showing of the horizon.

A large number of experiments can be done automatically (to test the performance of the algorithm). Select from the menu item <File
Experiments> and provide the experiment file.

1.1.4 Project Documentation

For a set of 3D points, this program constructs the 3D or 2D convex hull. The 3D CH construction code checks the input data, if all points lie in a same plane, it returns and the data is translate to the 2D CH construction program.
This project is developed in VC++6.0 environment.
To compile please include the header files in `JMath` and `JFileOp` directories.
To link please link with the library files in `JMath` and `JFileOp` directories and the library files for OpenGL `opengl32.lib glu32.lib glaux.lib`
The code for 3D CH construction includes :

- `JSinglyList.h /.cpp`, a singly linked list for storing the conflict graph mainly
- `JDCELVertex.h /.cpp`, a DCEL vertex
- `JDCELHalfEdge.h /.cpp`, a DCEL half edge
- `JDCELFacet.h /.cpp`, a DCEL facet
- `JConvexHull.h /.cpp`, for construction of the 3D CH

The code for 2D CH construction includes :

- `Coord3D.h /.cpp`, a vertex
- `LP3D.h /.cpp`, a list of 3D vertexes
- `hull2D.h /.cpp`, for construction of the 2D convex hull
- `Wrap2D,h /.cpp`, as there are compile & link problem when we try to integrate the code developed individually, this class is for solving the problem and for making a consistent interface as JConvexHull Other code are for providing a user interface and for maintain the flow of the program.

1.2 Outline of the Algorithm

The algorithms we use are those described in [1] Chapter 1 and Chapter 11. Nevertheless, the implementation of this algorithm on specific points needs to be detailed.

1.2.1 implementation of the DCEL

Our algorithm uses the *DCEL* structure defined in [1], but both *faces* and *vertices* contain the list $P_{conflict}$ and $F_{conflict}$ respectively.
In order to manage the memory and avoid any memory leak, a *vertex* x also contains a counter of the number of *half edges* whose origin is x.

1.2.2 Finding the *Horizon Border*

This algorithm is not describe in [1] and, as you will see, it can have an influence on the complexity of the algorithm. Our solution to find the *horizon border* is first to color every *half edge* in $F_{conflict}$ and using this coloration, the algorithm check every *half edge* until it find one on the horizon. To find the rest of the horizon, we just iteratively turn around each vertex of the horizon until we found the next *half edge* on the horizon as shown in Figure 1

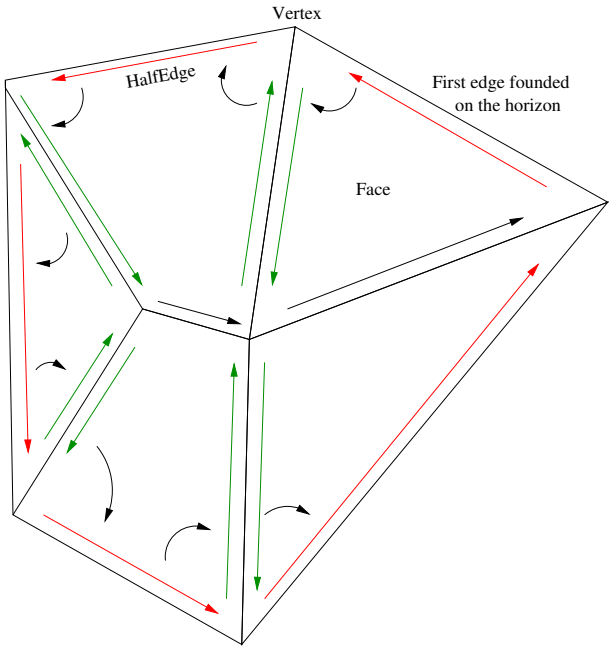

FIG. 1: Finding the rest of the horizon from the first edge

1.2.3 Handle Degenerate Case

The only degenerate case occurs when all the points are one the same plane, this case is detected during the construction of the first tetrahedron (first step of the algorithm) and this case is handled by the 2D convex hull algorithm.

1.3 Complexity

The expected running time of this algorithm has been proved to be $O(n.log(n))$ in [1], section 11.3. We will not re-write the whole demonstration here, but just clarify some points as finding the horizon border, the worst case complexity and the complexity in a special case.

1.3.1 Finding the Horizon Border

The demonstration about the complexity assumes that finding the *horizon border* takes linear time in $card(F_{conflict}$, but is it the case with our algorithm ?
In fact, $F_{conflict}$ can be seen as a planar graph, EULER's formula thus implies the number of edges is bounded : $O(Nb_{edges} \leq F_{conflict})$. Our algorithm first colors all edges of $F_{conflict}$, this takes $O(F_{conflict})$, and after, it scans the edges of $F_{conflict}$ a first time to find a first edge on the horizon and scans a second time the edges to find the rest of the horizon (in fact, this algorithm does not check every edge two times), so it also takes $O(F_{conflict})$. This way, our algorithm return the horizon border in $O(F_{conflict})$.

1.3.2 Using the *Framework for Randomized algorithms*

[1] In the demonstration of the complexity provided by [1], section 11.3, the results on the *Framework for Randomized algorithms* are used on a configuration space to bound the value of $\sum_e card(P(e))$. Nevertheless, the configuration space has not still be proved to be suitable.

Configuration Space :
According to [1] section 11.3, we use a configuration space (S, Π, D, K) where a flap $\Delta \in \Pi$ is defined as an ordered 4-uple of points (p, qp, s, t) that do not lie in a plane. The defining set $D(\Delta)$ is simply the set p, q, s, t and a point $x \in S$ is in $K(\Delta)$ if and only if it lies in one of the following regions :

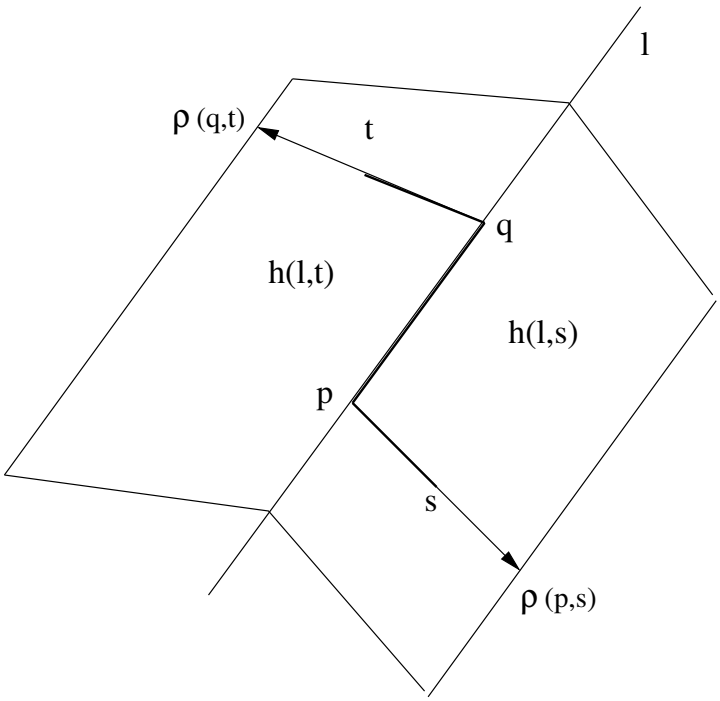

FIG. 2:

- 1. outside the closed 3D convex wedge defined by $h(l,t)$ and $h(l,s)$
- 2. inside $h(l,s)$ but outside the closed 2D wedge defined by $\rho(p,q)$ and $\rho(p,s)$
- 3. inside $h(l,t)$ but outside the closed 2D wedge defined by $\rho(p,q)$ and $\rho(q,t)$
- 4. inside the line l but outside the segment (pq)
- 5. inside the half-line $\rho(p,s)$ but outside the segment (ps)
- 6. inside the half-line $\rho(q,t)$ but outside the segment (qt)

Finally, $\tau(S) = \Delta \in \Pi \mid D(\Delta) \in S \; and \; K(\Delta) \cap S =$

[1] cf [1], section 9.5

Lemma (lemma 11.5 of the [1], section 11.3) : A flap $\Delta = (p, q, s, t)$ is in $\tau(S)$ if and only if $\overrightarrow{pq}$, $\overrightarrow{ps}$ and $\overrightarrow{qt}$ are edges of the convex hull $\mathscr{CH}(S)$, there is a facet f_1 incident to $\overrightarrow{pq}$ and $\overrightarrow{ps}$, and a different facet f_2 incident to $\overrightarrow{pq}$ and $\overrightarrow{qt}$. Furthermore, if one of the facets f_1 or f_2 is visible from a point $x \in P$ then $x \in K(\Delta)$.

demonstration :

=> : Let $\Delta = p, q, s, t$ be a configuration such that $\Delta \in \tau(S)$
We first prove that every points of $D(\Delta)$ is a vertex of $\mathscr{CH}(S)$: Assume (for example) that t is not a vertex ; then if t lies strictly inside the convex hull, it is quite easy to find a vertex x which lie outside the convex 3D wedge defined by $h(l, s)$ and $h(l, t)$ (cf 1.) ; if t lies on an edge of the convex hull, it is immediate that a vertex $x$on the half line (qt) but outside the segment exits (cf 6.) ; and finally, if t lies strictly on a face of the convex hull, then it's also immediate that a vertex inside $h(l, t)$ but outside the closed 2D wedge defined by $\rho(q, t)$ and $\rho(p, q)$ (cf 2.). Thus, in each case we can find a point x such that $x \in D(\Delta) \cap K(\Delta)$, then $D(\Delta) \cap K(\Delta) \neq$ and so $\Delta \notin \tau(S)$ which leads to a contradiction. In the same way we can prove that every point of $D\Delta)$ is a vertex of $\mathscr{CH}(S)$.

We now prove the 3 segment (qt), (pq), (ps) are edges of $\mathscr{CH}(S)$: Assume (for example) that (qt) is not an edge of the convex hull ; then (qt) lies inside the convex hull and since t is a vertex, there is an edge of the convex hull which start from t to a vertex x such that x is outside the convex 3D wedge defined by $h(l, s)$ and $h(l, t)$ (cf 1.). Thus $x \in D(\Delta) \cap K(\Delta)$, then $D(\Delta) \cap K(\Delta) \neq$ and so $\Delta \notin \tau(S)$ which leads to a contradiction. In the same way, we can prove that (pq) and (ps) are edges of $\mathscr{CH}(S)$.

<= : Let (pq) be an edge of $\mathscr{CH}(S)$ and p, q, s, t defined as in the Figure 2 ; then it is obvious the flap $\Delta = (p, q, s, t)$ is in $\tau(S)$.

1.3.3 Worst Case Complexity

Theorem : The worst case running time of this algorithm is $O(n^2)$

demonstration : Still not solved <

Example :
We now can show an example of input configuration for which, the algorithm effectively runs in $O(n^2)$

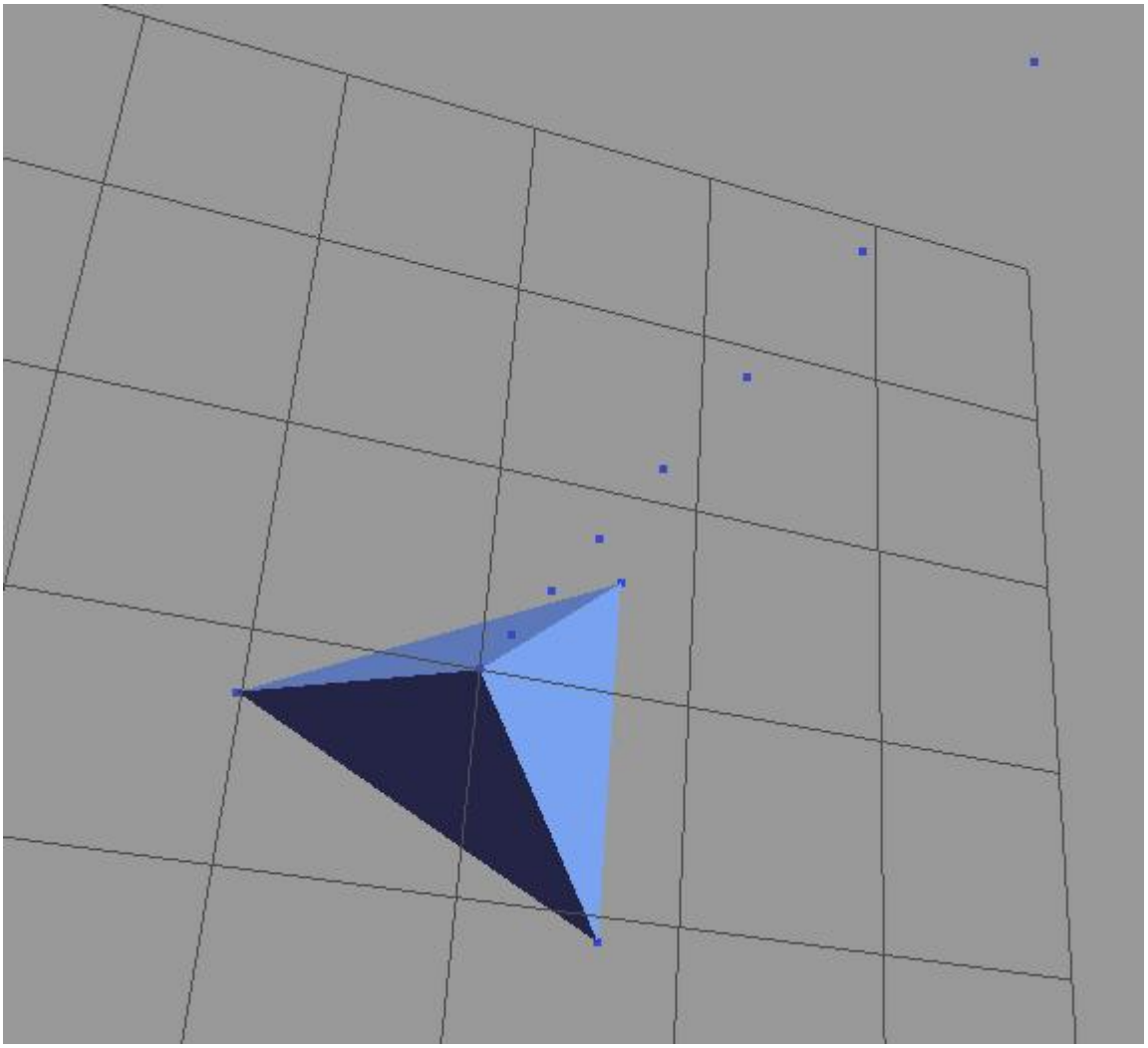

FIG. 3: Worst Case Configuration

For n, number of vertices, N_f the number of created faces during the running time, N_d the number of deleted faces and the value which determines the running time, $\sum_e P(e)$:

n	N_f	N_d	$\sum_e P(e)$
10003	$300001 \cong 3.n$	$29997 \cong 3.n$	$149955003 \cong n^2$

1.3.4 Special Input

In this part we will prove that under certain conditions on the input, the algorithm can run faster.
If the set of points P has been created by uniformly picking points at random in a ball, then we assume that the number of extreme points in a random sample of P of size r is only $O(r^\alpha$ for some constant $\alpha < 1$. This result has been proved in [4].

Theorem : Assume the input of the algorithm is a set S with $card(S) = n$.
If the number of extreme points in a random sample $P \in S$ of size r is only $O(r^\alpha)$ for some constant $\alpha < 1$, then the running time of the algorithm is $O(n)$

demonstration : We know from the demonstration of the complexity of this algorithm in [1] (section 11.3), to bound the total expected running time, we have to bound the expected value of :

$$\sum_e card(P(e))$$

Where the summation is over all horizon edges that appear at any stage of the algorithm and $P(e) = P_{conflict}(f_1) \cup P_{conflict}(f_2)$ where f_1 and f_2 are the facets incident to the edge e in the old convex hull.
Form the analysis using the *Framework for randomized algorithms*[2], it follows :

$$\begin{aligned} \sum_e card(P(e)) &\leq \sum_\Delta card(K(\Delta)) \\ &\leq \sum_{r=1}^{n} 16. \left(\frac{n-r}{r}\right) . \left(\frac{E[card(\tau(P_r))]}{r}\right) \end{aligned} \tag{1}$$

but, $card(\tau(P_r))$ represent the number of flaps in $\mathscr{CH}(P_r)$, since there is two flaps per edge and $O(n_{edge}) = O(n_{vertex})$, then $card(\tau(P_r)) = O(n_{vertex}) = O(r^\alpha)$; thus :

$$\begin{aligned} \sum_e card(P(e)) &\leq \sum_{r=1}^{n} 16. \left(\frac{n-r}{r}\right) . \left(\frac{O(r^\alpha)}{r}\right) \\ &= O\left(\sum_{r=1}^{n} \frac{n-r}{r^{2-\alpha}}\right) \end{aligned}$$

Because $f(t) = \frac{n-t}{t^{2-\alpha}}$ decreases on $[0, n]$, by theorem we thus can frame the sum in the following way :

$$\int_1^n \frac{n-r}{r^{2-\alpha}}.dr \leq \sum_{r=1}^{n} \frac{n-r}{r^{2-\alpha}} \leq (n-1) + \int_1^{n-1} \frac{n-r}{r^{2-\alpha}}.dr$$

then

$$O\left(\sum_{r=1}^{n} \frac{n-r}{r^{2-\alpha}}\right) = O(n)$$

and this way $\boxed{\sum_e card(P(e)) = O(n)}$

A bound for the complexity of the algorithm with such an input therefore is $O(n)$.

[2] [1], section 9.5

2 Performance Analysis

2.1 Worst Case Analysis

We already know that the worst case configuration makes our algorithm run in $O(n.ln(n))$, but what if we try a permutation on the order of the input points ? In fact, the expected running time of this configuration, if the points are randomly handled is linear !

Lemma : For the current configuration, with a random permutation on the input points, the running time of our algorithm is linear.

demonstration : Obviously, the number of extreme points of such a configuration is a constant (4). Thus we can re-use the demonstration in 1.3.4, but this time, $E[card(\tau(P_r))] = O(1)$. Then :

$$\begin{aligned}
\sum_e card(P(e)) &\leq \sum_\Delta card(K(\Delta)) \\
&\leq \sum_{r=1}^{n} 16. \left(\frac{n-r}{r}\right) . \left(\frac{E[card(\tau(P_r))]}{r}\right) \\
&\leq \sum_{r=1}^{n} 16.(\frac{n-r}{r}).(\frac{O(1)}{r}) \\
&= O(\sum_{r=1}^{n} \frac{n-r}{r^2}) \\
&= O(n)
\end{aligned}$$

2.2 Experiment on the Special Input of 1.3.4

We recall that if we pick the points at random, uniformly in a d-dimensional hyper-cube, the the expected number of extreme points is $O(log^{d-1}(n))$ [4] and thus, for our case, $O(log_3(n)) = O(r^\alpha)$ $(\alpha < 1)$ gives an expected running time of our algorithm in $O(n)$ theoretically. We now will verify this result through experiment.

Experimentation Conditions : We are using a good random number generator to generate both sets of points randomly chosen uniformly in a cube and sets of points which coordinates follow a Gaussian distribution. This generator is the one of L'Ecuyer with Bays-Durham shuffle, found in Numerical Recipes (`http ://www.nr.com`) (We did not implement this PRNG by ourselves).
To measure the running time, we do no measure the time our process is within the CPU nor the time required for memory allocations, we just count $\sum_e card(P(e))$, which was much easier for us to set up. This way, our results might not be as rigorous as they should be, but what interested us is the method for the analysis.

Experiment parameters : As input of our algorithm, we used a set of points, randomly chosen uniformly in a cube and we plot $running\,times = f(n)$ (n, number of points in the input). We ran our algorithm for n from 1000 to 100000 with a step of 1000 and for each step 10 times. Here the results on the complexity Figure 4.

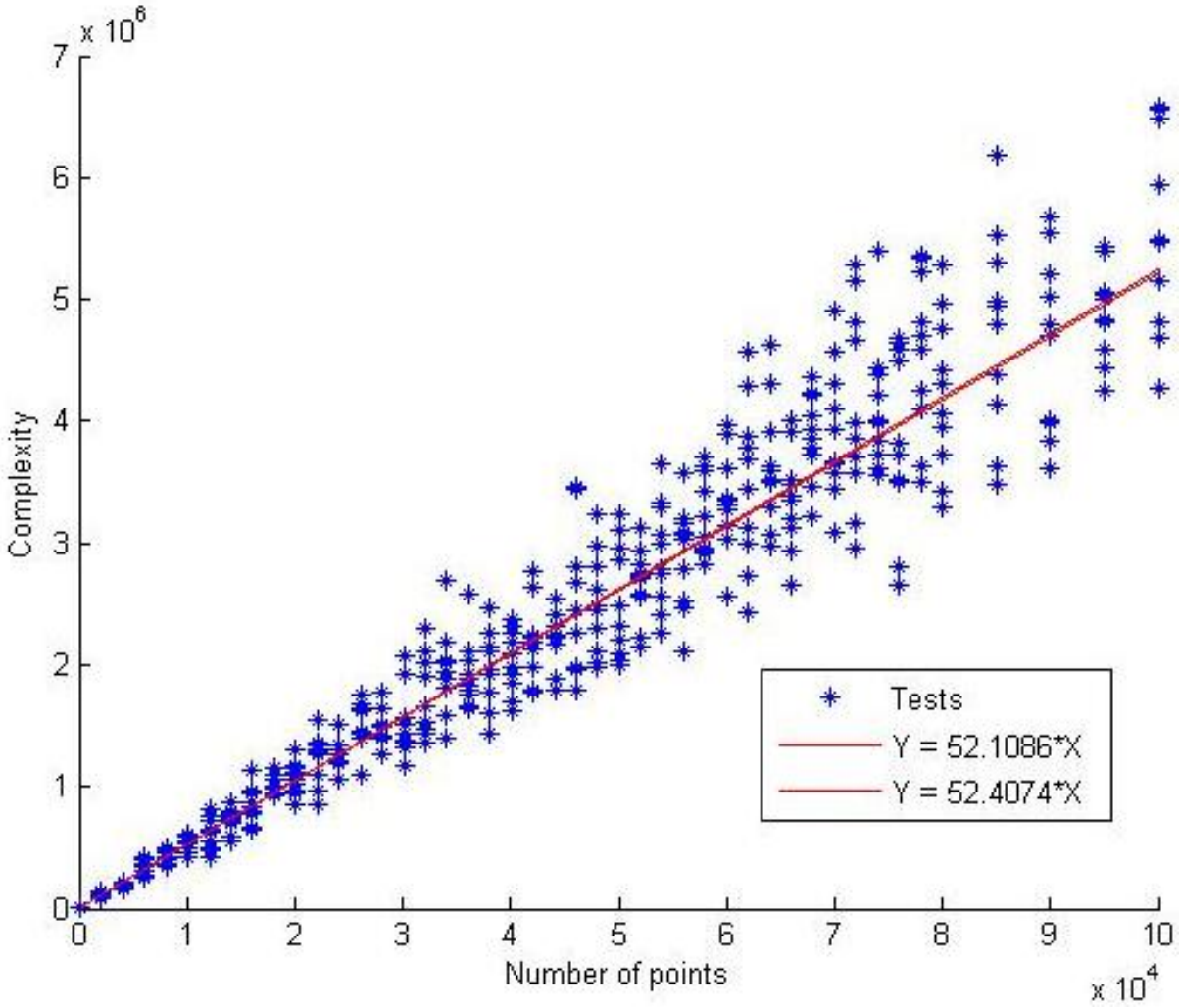

FIG. 4: $Running\,Time = f(n)$

2.2.1 Estimation

Now, from a set of $n \simeq 10000$ tests on set of point of *the same cardinality*, we want to calculate an estimation of the expected running time m and the variance (σ^2) of the complexity of the algorithm with such an input ; here, we will find an interval I such that

$$P\left[m \in I\right] = 0.95$$

and an interval J such that

$$P\left[\sigma \in J\right] = 0.95$$

Notation :
We perform n experiments and we call X_i be the random variable which give the complexity ($\in \mathbb{N}$) of the i^{st} test. i.e. X measurable on the probabilist space $(\Omega, \mathscr{F}, P)$

$$X : (\Omega, \mathscr{F}, P) \to \mathbb{N}$$

We call x_i the value of the complexity given by our i^{st} experiment.
We define $\overline{X}$ such that

$$\overline{X} = \frac{1}{n}.\sum_{i=1}^{n} X_i$$

We define S, an estimator without bias of $\sigma = \sqrt{VAR[X]}$

$$S^2 = \frac{1}{n-1}.\sum_{i=1}^{n}(X_i - \overline{X})^2$$

We also define $\overline{x}$ and s such that :

$$\overline{x} = \frac{1}{n}.\sum_{i=1}^{n} X_i$$

$$s^2 = \frac{1}{n-1}.\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Lemma 1 : $\overline{X}$ follows a Gauss law ($\mathscr{N}(m, \sigma^2)$)

***demonstration* :** Let i randomly chosen.
X_i takes its values in $\mathbb{N}$ and we know the worst case complexity is $O(n^2)$, thus $X \in B$ where B is a bounded interval of $\mathbb{N}$. Then, obviously, X^2 is integrable on $(\Omega, \mathscr{F}, P)$; each test is realised independently and follows the same law of probability as the others. Under those hypothesis, we can apply the theorem of central limit and say the serial $\overline{X}$ converges in law to a random variable which follows the normal law ($\mathscr{N}(m, \sigma^2)$).
As in our case, n is high ($n \simeq 10000$), we can assume that $\overline{X}$ follows the normal law (or Gauss' law).

We now define X^* :

$$X^* = \sqrt{n}.\frac{\overline{X} - m}{S}$$

Lemma 2 : X^* follows a Student law of degree $n-1$

Note : Density of the Student law of degree n

$$f(x) = \frac{1}{\sqrt{n.\pi}}.\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)}.\left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}}$$

with

$$\Gamma(s) = \int_0^{\infty} t^{s-1}.e^t.dt$$

demonstration **:** We there assume X follows a normal law. The graphs Figure 5 and Figure 6 show that our assumption is not unreasonable since we obtain a *Gaussian* curve :
Figure 5 and Figure 6 represent the number of tests on sets of respectively 1000 and 5000 points which the complexity is within an interval respectively 350 and 2600 large.

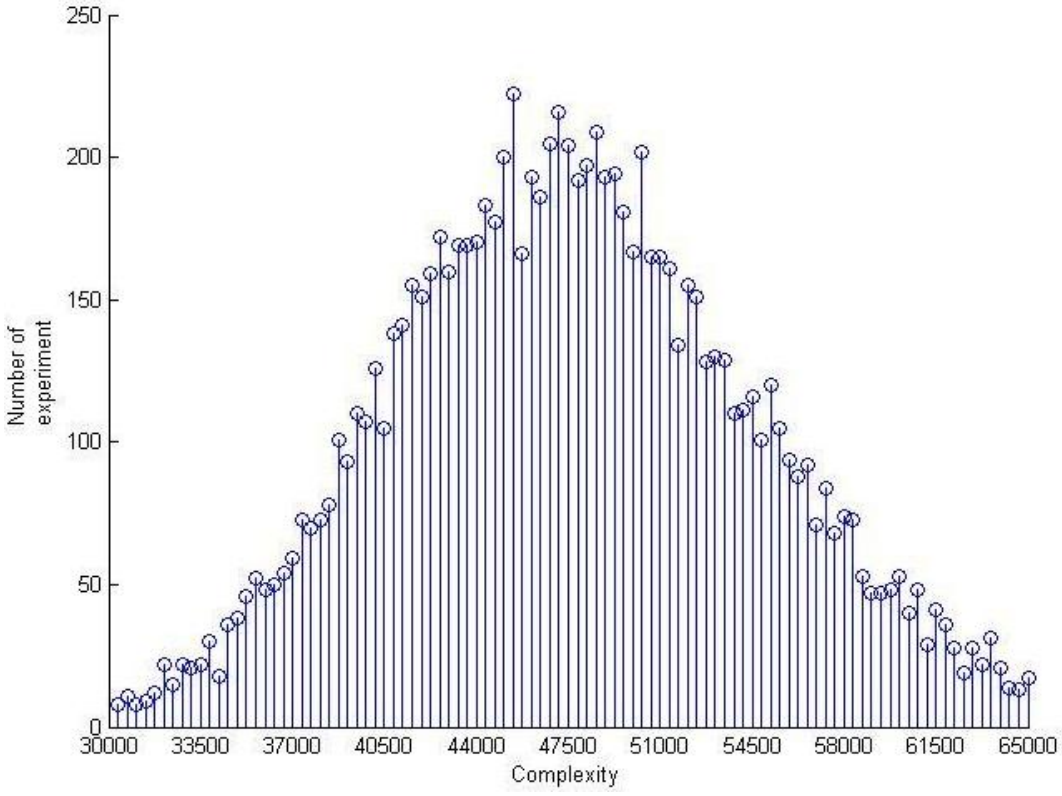

FIG. 5:

As X follows a normal law, then S follows a law of χ^2_{n-1}.
From theorem, as $\overline{X}$ follows a normal law and $(n-1).\frac{S^2}{\sigma^2}$ follows a law of χ^2_{n-1}, then $X^* = \sqrt{n}.\frac{\overline{X}-m}{S}$ follows a STUDENT law of degree $n-1$.

Estimation of the Expected Running Time with n tests :
Now, we will be able to give an interval I such that (for a given α)

$$P\left[m \in I\right] = 1-\alpha$$

We know, $X^* = T_{n-1}$ follows a STUDENT law. let t_β such that $P[T_{n-1} \leq t_\beta] = 1-\beta$.
As the density of a STUDENT law is symmetric, then $t_{1-\alpha/2} = -t_{\alpha/2}$ and thus

$$P\left[-t_{\alpha/2} \leq \sqrt{n}.\frac{\overline{X}-m}{S} \leq t_{\alpha/2}\right] = 1-\alpha$$

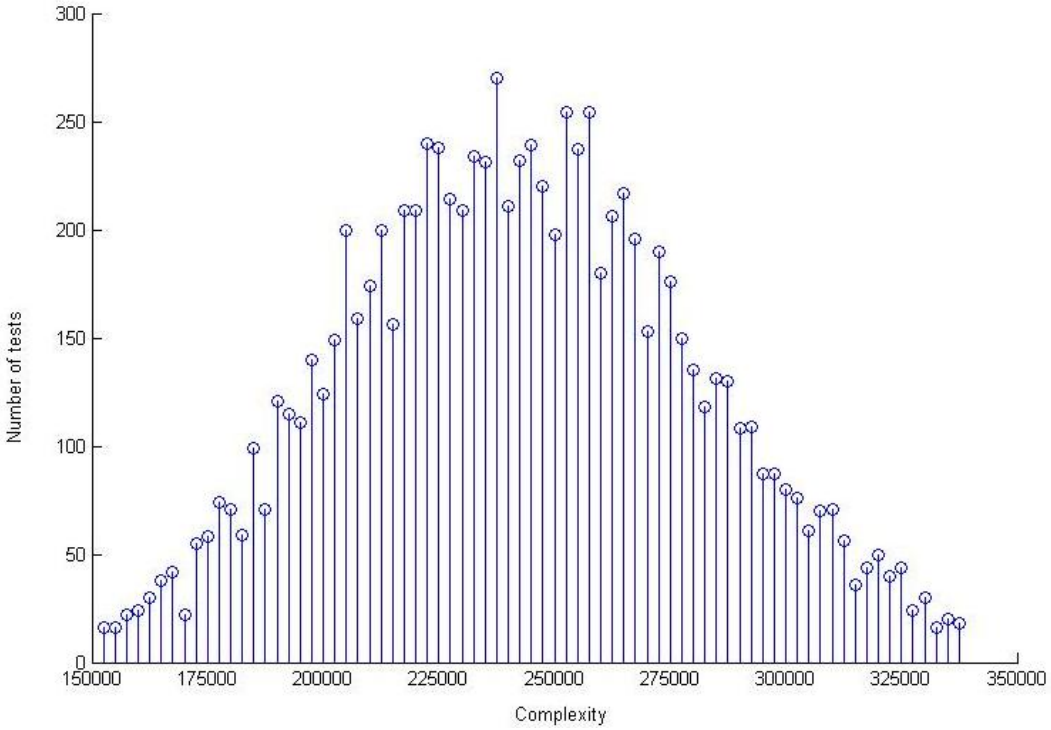

FIG. 6:

This way, for the realisation $\overline{x}$ of $\overline{X}$ and s^2 of S

$$I(\overline{x}, s^2) = \left]\overline{x} - \frac{s}{\sqrt{n}}.t_{\alpha/2}, \overline{x} + \frac{s}{\sqrt{n}}.t_{\alpha/2}\right[$$

Is our interval I

Results for tests on sets of 1000 points :
In this experiment, $n = 10000$, $\overline{x} = 47723$ and $s^2 = 5,2681.10^7$. We have to find suitable $t_{\alpha/2}$ such as $P[T_{n-1}] = 1 - \alpha/2$, we can read in the table given in ANNEXE, $t_{\alpha/2} = 1,960$ for $\alpha = 0,05$. Thus :

$I(\overline{x}, s^2) =]47580, 47865[$

Results for tests on sets of 5000 points :
In this experiment, $n = 9511$, $\overline{x} = 261290$ and $s = 37170$.

$I(\overline{x}, s^2) =]260543, 262037[$

Estimation of the variation of the running time with n tests :
We still assume X follows a normal law such that $(n-1).\frac{S^2}{\sigma^2}$ follows a χ^2_{n-1} law.

Note : Density of a χ^2_{n-1} law

$$f(x) = \frac{1}{2^{\frac{n-1}{2}}.\Gamma\left(\frac{n-1}{2}\right)}.x^{\frac{n-3}{2}}.e^{-\frac{x}{2}}.\mathbf{1}_{]0,+\infty[}$$

We define x_β such that $P[(n-1).\frac{S^2}{\sigma^2} \leq x_\beta] = 1-\beta$, then

$$P\left[-x_{1-\beta/2} \leq (n-1).\frac{S^2}{\sigma^2} \leq x_{\beta/2}\right] = 1-\beta$$

And finally, for the realisation s^2 of S^2, we obtain the interval

$$J = \left]\frac{(n-1).s^2}{x_{\beta/2}}, \frac{(n-1).s^2}{x_{1-\beta/2}}\right[$$

Note : we can not give any result because we did not find the table for the χ^2 law (as the table for STUDENT law given in Annexe) and we have no time to calculate it...

2.3 Experiment on Another Distribution

Here we we use sets of points with a GAUSS' distribution on each coordinate. We use the same PRNG as before and make the same experiments.

First Idea about the Complexity We plot $running\,times = f(n)$ (n, number of points in the input). We ran our algorithm for n from 0 to 100000 with a step of 5000 and for each step 20 times. Here the results on the complexity Figure 7.

From what we can see on Figure 7, the complexity is still linear, but we did not prove it. We maybe could prove the number of extreme points is very small, and we could also maybe prove by statistical analysis that the expected complexity is linear ; but we did not have enough time to do this.

Estimation on the expected complexity and its variation Let now assume the complexity is linear, we want to know precisely the coefficient of linearity and we want to know if the variation is important. According to the Figure 8, we still assume that X follows GAUSS' law, thus all the results above remains[3].

Figure 8 represents the number of tests on sets of respectively 5000 points which the complexity is within an interval 2666 large.

[3] If X does not follow a GAUSS'law, just the results about expected complexity remains, the estimation on the variation is false in this case

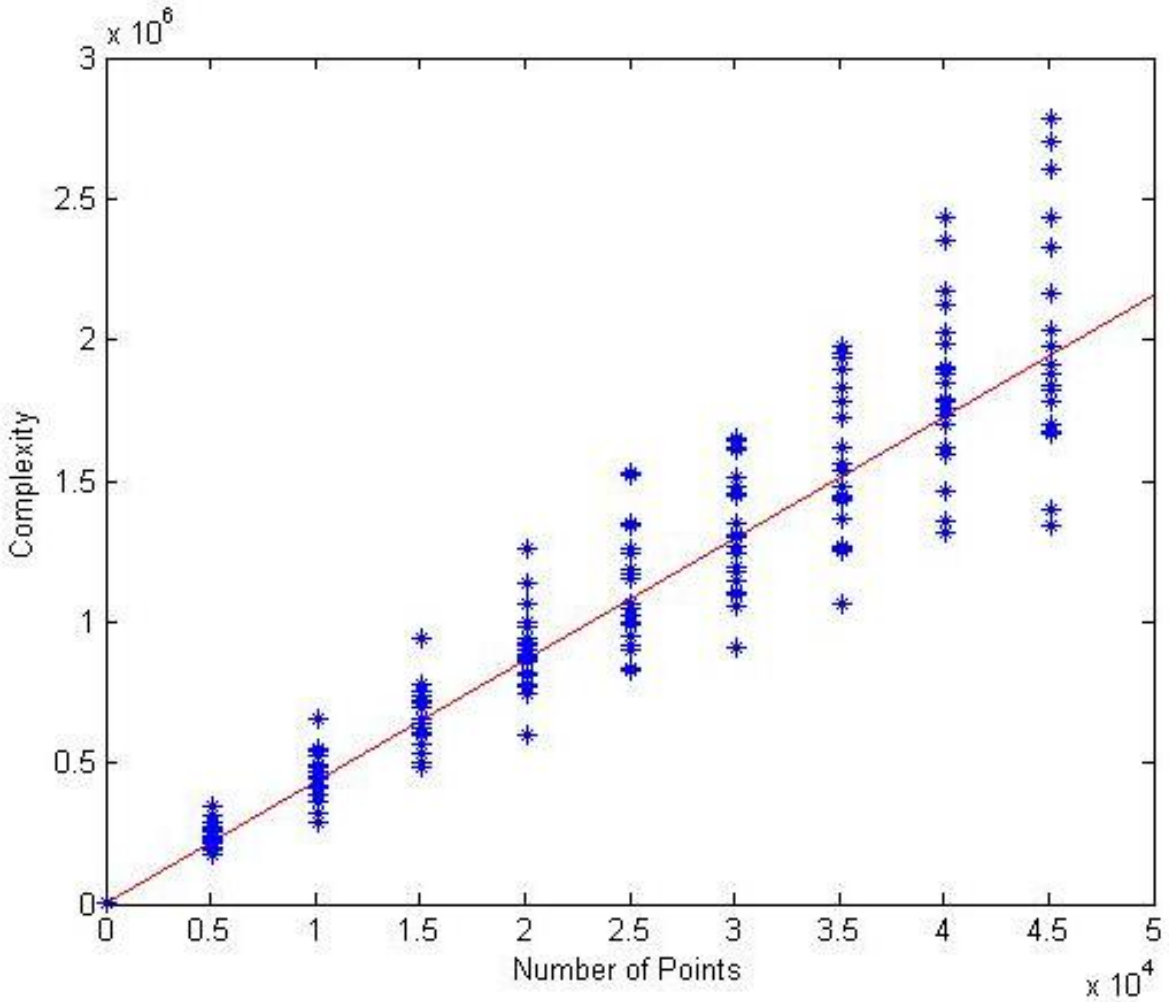

FIG. 7: $Running Time = f(n)$

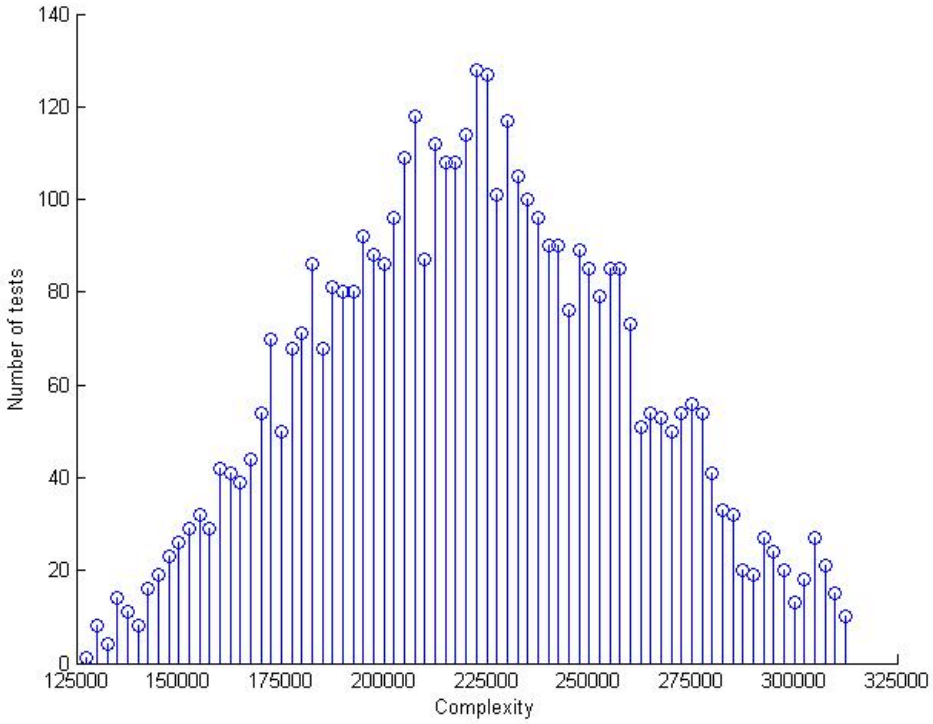
Number of tests
Complexity
0
20
40
60
80
100
120
140
125000
150000
175000
200000
225000
250000
275000
300000
325000

FIG. 8:

this way,

$$I(\overline{x}, s^2) = \left]\overline{x} - \frac{s}{\sqrt{n}}.t_{\alpha/2}, \overline{x} + \frac{s}{\sqrt{n}}.t_{\alpha/2}\right[$$

and

$$J = \left]\frac{(n-1).s^2}{x_{\beta/2}}, \frac{(n-1).s^2}{x_{1-\beta/2}}\right[$$

then

$$I(\overline{x}, s^2) =]214397, 216663[$$

Conclusion

We could have been further into the performance analysis, proving the linear expected complexity for the 2.3 instead of assuming it, proving the variation is linear with the number of points in input etc...
Nevertheless, we already understand the power of a statistical analysis to give rigorous, precise and useful results.
We could have improved this study, but we did not have enough time to perform a better analysis because computing a convex hull of more than 50000 points takes a long time on our computer because of the lack of memory and we did not measure the complexity from the CPU time which is the most important thing which could dodge the issues of our performance analysis.

Bibliographie

[1] M. de Berg M. van Kreveld M. Overmars O. Shwarzkopf. *Computational Geometry Algorithms and Applications*. Springer, 1997.

[2] J. F. Le Gall. *Cours d'integration et probabilités*. Ecole Normale Superieure de Paris, 2003.

[3] Melle Rouy. Probabilités et statistiques, première année. *http ://www.mi.ec-lyon.fr*, 2003.

[4] J. L. Bentley H. T. Kung M. Schkolnick and C. D. Thomson. On the average number of maxima in a set of vectors. *ACM*, 1978.

Annexes

La fonction de répartition de la loi de Student

$n \mid P$	0,60	0,70	0,80	0,90	0,95	0,975	0,990	0,995	0,999	0,9995
1	0,325	0,727	1,376	3,078	6,314	12,71	31,82	63,66	318,3	636,6
2	0,289	0,617	1,061	1,886	2,920	4,303	6,965	9,925	22,33	31,60
3	0,277	0,584	0,978	1,638	2,353	3,182	4,541	5,841	10,22	12,94
4	0,271	0,569	0,941	1,533	2,132	2,776	3,747	4,604	7,173	8,610
5	0,267	0,559	0,920	1,476	2,015	2,571	3,365	4,032	5,893	6,859
6	0,265	0,553	0,906	1,440	1,943	2,447	3,143	3,707	5,208	5,959
7	0,263	0,549	0,896	1,415	1,895	2,365	2,998	3,499	4,785	5,405
8	0,262	0,546	0,889	1,397	1,860	2,306	2,896	3,355	4,501	5,041
9	0,261	0,543	0,883	1,383	1,833	2,262	2,821	3,250	4,297	4,781
10	0,260	0,542	0,879	1,372	1,812	2,228	2,764	3,169	4,144	4,587
11	0,260	0,540	0,876	1,363	1,796	2,201	2,718	3,106	4,025	4,437
12	0,259	0,539	0,873	1,356	1,782	2,179	2,681	3,055	3,930	4,318
13	0,259	0,538	0,870	1,350	1,771	2,160	2,650	3,012	3,852	4,221
14	0,258	0,537	0,868	1,345	1,761	2,145	2,624	2,977	3,787	4,140
15	0,258	0,536	0,866	1,341	1,753	2,131	2,602	2,947	3,733	4,073
16	0,258	0,535	0,865	1,337	1,746	2,120	2,583	2,921	3,686	4,015
17	0,257	0,534	0,863	1,333	1,740	2,110	2,567	2,898	3,646	3,965
18	0,257	0,534	0,862	1,330	1,734	2,101	2,552	2,878	3,611	3,922
19	0,257	0,533	0,861	1,328	1,729	2,093	2,539	2,861	3,579	3,883
20	0,257	0,533	0,860	1,325	1,725	2,086	2,528	2,845	3,552	3,850
21	0,257	0,532	0,859	1,323	1,721	2,080	2,518	2,831	3,527	3,819
22	0,256	0,532	0,858	1,321	1,717	2,074	2,508	2,819	3,505	3,792
23	0,256	0,532	0,858	1,319	1,714	2,069	2,500	2,807	3,485	3,767
24	0,256	0,531	0,857	1,318	1,711	2,064	2,492	2,797	3,467	3,745
25	0,256	0,531	0,856	1,316	1,708	2,060	2,485	2,787	3,450	3,725
26	0,256	0,531	0,856	1,315	1,706	2,056	2,479	2,779	3,435	3,707
27	0,256	0,531	0,855	1,314	1,703	2,052	2,473	2,771	3,421	3,690
28	0,256	0,530	0,855	1,313	1,701	2,048	2,467	2,763	3,408	3,674
29	0,256	0,530	0,854	1,311	1,699	2,045	2,462	2,756	3,396	3,659
30	0,256	0,530	0,854	1,310	1,697	2,042	2,457	2,750	3,385	3,646
32	0,256	0,530	0,853	1,309	1,694	2,037	2,449	2,738	3,365	3,622
34	0,255	0,529	0,852	1,307	1,691	2,032	2,441	2,728	3,348	3,601
36	0,255	0,529	0,852	1,306	1,688	2,028	2,434	2,719	3,333	3,582
38	0,255	0,529	0,851	1,304	1,686	2,024	2,429	2,712	3,319	3,566
40	0,255	0,529	0,851	1,303	1,684	2,021	2,423	2,704	3,307	3,551
50	0,255	0,528	0,849	1,298	1,676	2,009	2,403	2,678	3,261	3,496
60	0,254	0,527	0,848	1,296	1,671	2,000	2,390	2,660	3,232	3,460
70	0,454	0,527	0,847	1,294	1,667	1,994	2,381	2,648	3,211	3,435
80	0,254	0,527	0,846	1,292	1,664	1,990	2,374	2,639	3,195	3,415
90	0,254	0,526	0,846	1,291	1,662	1,987	2,368	2,632	3,183	3,402
100	0,254	0,526	0,845	1,290	1,660	1,984	2,365	2,626	3,174	3,389
200	0,254	0,525	0,843	1,286	1,653	1,972	2,345	2,601	3,131	3,339
500	0,253	0,525	0,842	1,283	1,648	1,965	2,334	2,586	3,106	3,310
∞	0,253	0,524	0,842	1,282	1,645	1,960	2,326	2,576	3,090	3,291

Soit X une variable aléatoire qui suit une loi de Student à n degrés de liberté, notée $X \sim t_n$.
La table donne les valeurs $t_{n;P}$ telles que

$$F_X(t_{n;P}) \stackrel{\text{déf}}{=} \mathbb{P}(X \leq t_{n;P}) = P.$$

Par exemple: $t_{4;0,90} = 1,533$ ce qui implique que $\mathbb{P}(X \leq 1,533) = 0,90$ si $X \sim t_4$.