Reducing the number of points on the convex hull calculation using the polar space subdivision in E^2

Vaclav Skala, Michal Smolik, Zuzana Majdisova
Department of Computer Science and Engineering, Faculty of Applied Sciences
University of West Bohemia
Plzen, Czech Republic
Email: {skala, smolik, majdisz}@kiv.zcu.cz

***Abstract*—A convex hull of points in E^2 is used in many applications. In spite of low computational complexity $O(h \log n)$ it takes considerable time if large data processing is needed. We present a new algorithm to speed-up any planar convex hull calculation. It is based on a polar space subdivision and speed-up known convex hull algorithms of $3,7$ times and more. The algorithm estimates the central point using 10% of the data; this point is taken as the origin for the polar subdivision. The space subdivision enables a fast and very efficient reduction of the given points, which cannot contribute to the final convex hull. The proposed algorithm iteratively approximates the convex hull, leaving only a small number of points for the final processing, which is performed using a "standard" algorithm. Non-eliminated points are then processed by a selected standard convex hull algorithm. The algorithm is simple and easy to implement. Experiments proved numerical robustness as well.**

***Keywords*—Convex hull; iterative approximation; space subdivision; reduction of points;**

I. Introduction

A convex hull was one of the first sophisticated geometry algorithms to be computed and there are many variations of it. The most common form of this algorithm involves the determination of the smallest convex set, called the "convex hull", containing a discrete set of points. There are numerous applications for convex hulls: collision avoidance, maximum distance using convex hull diameter for large data sets (Skala, 2013), (Skala and Majdisova, 2015), hidden object determination, and shape analysis, point inside polygon (Skala and Smolik, 2015), to name a few.

A subset $S \subseteq \mathbb{R}^2$ is convex if and only if for any two points $\boldsymbol{p}, \boldsymbol{q} \in S$ the line segment with endpoints $\boldsymbol{p}$ and $\boldsymbol{q}$ is completely contained in S. The convex hull $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S. The convex hull of a set of points P is a convex polygon with vertices in P.

A. Time Complexity

The search for the fastest algorithm is a pursuit which has been preoccupying many authors for many years now. Many have found excellent algorithms. Lately performance reached $O(n \log h)$ complexity where h is the number of points forming the convex hull and n is the number of input points.

There are many well-known algorithms used for the calculation of convex hull in $2D$. A comparison of the time complexity for some of them can be seen in Table 1. (Graham, 1972) gave a complex hull algorithm with $O(n \log n)$, the worst-case running time. Later it was shown that, in any model of computation where sorting has an $O(n \log n)$ lower bound, every convex hull algorithm must require $O(n \log n)$ time for some inputs. Despite these matching upper and lower bounds, and probably due to the many applications of convex hulls, a number of other planar convex hull algorithms have been published since Graham's algorithm. Moreover, the Marriage before Conquest algorithm (Kirkpatrick and Seidel, 1986) computes the convex hull in $O(n \log h)$ time, where h is the number of vertices of the final convex hull. The same authors showed that, on algebraic decision trees of any fixed order, $O(n \log h)$ is a lower bound for computing convex hulls of a set of n points, having a convex hull with h vertices.

Table 1. Comparison of 2D convex hull algorithms and their time complexity. The number of input points is n and h is the number of vertices of the final convex hull. Note that $h < n$, so $nh < n^2$.

Algorithm	Expected time complexity	Reference
Gift Wrapping	$O(nh)$	(Chand and Kapur, 1970)
Graham Scan	$O(n \log n)$	(Graham, 1972)
Jarvis March	$O(nh)$	(Jarvis, 1973)
QuickHull	$O(nh)$	(Eddy, 1977), (Bykat, 1978)
Divide & Conquer	$O(n \log n)$	(Preparata and Hong, 1977)
Monotone Chain	$O(n \log n)$	(Andrew, 1979)
Incremental	$O(n \log n)$	(Kallay, 1984)
Marriage before Conquest	$O(n \log h)$	(Kirkpatrick and Seidel, 1986)
Chan's algorithm	$O(n \log h)$	(Chan, 1996)
Ordered hull	$O(n \log h)$	(Liu and Chen, 2007)

II. Proposed Algorithm

In the following section, we will introduce a new approach to speeding any convex hull computation in E^2. The main idea of this algorithm is to discard as many points as possible before the actual convex hull calculation takes place. The technique used in this algorithm is based on division of space into several polar sectors.

In section 2.A, we will show the first step of the algorithm proposed which is the location of points inside the created initial polygon. In section 2.B, we will show how to divide points into polar shaped sectors. In section 2.C, we will show how to reduce the suspicious points. And finally, in section 2.D, we will show a algorithm for calculation of the convex hull from the selected divided points.

A. Location of Points inside Polygon

An important property of input points is that the most extreme point on any axis is part of a convex hull. This fact is apparent if we consider an example in which an extreme point is not in the convex hull. This would mean the perimeter of

the convex hull passes through a point less extreme than that point. However, it is then obvious that the extreme point would not be included in the enclosed set, thus breaking a fundamental characteristic of convex hulls. This property is used in our algorithm for speeding-up the creation of the convex hull.

The first step is finding the axis aligned bounding box (AABB) of the input points, which is of $O(n)$ time complexity. For the future described polar space subdivision we do not need the exact extremal point, close enough extremal points are sufficient for our purpose. Therefore we do not have to search for them through all the input points, i.e. we can search only approximately 10% of all points. This simplification can save us some time and will not cause any disadvantage in future calculations. So we get four distinct points or, in some cases, only three or even just two.

Using these points we can create a convex polygon, see Fig. 1. One important property of this polygon is that any of the points lying inside cannot be the points of the final convex hull. Keeping this in mind, we can create the initial test for determining whether the point is inside/outside the polygon and dismiss a lot of points by using this easy initial test.

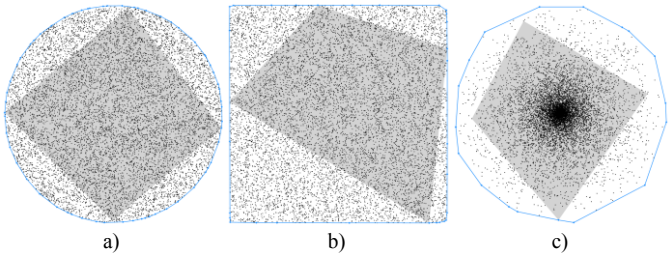


Fig. 1. Location of initial inner testing polygon inside the convex hull for 10^4 points (10% of points used for finding AABB box): a) uniform points in circle, b) uniform points in square, c) Gauss points.

The location test of a point inside a polygon can be determined using the following steps. Each side of the polygon is an oriented line and thus we can calculate $F_i(\boldsymbol{x})$, where $\boldsymbol{x}$ is the point and $F_i(\boldsymbol{x}) = 0$ is the implicit formula for a side with index i:

$$F_i(\boldsymbol{x}) = a_i x + b_i y + c_i = 0\,. \tag{1}$$

If $F_i(\boldsymbol{x}) < 0$ for at least one $i \in \{1, \dots, 4\}$ then the point $\boldsymbol{x}$ lies outside of the polygon and has to be used for further processing. Thus if for all $i \in \{1, \dots, 4\}$ is $F_i(\boldsymbol{x}) \geq 0$ then the point $\boldsymbol{x}$ lies inside of the polygon and can be discarded.

B. Division of Points into Sectors

Some points were discarded because of their location inside the inner polygon. Other points have to be further processed. The $2D$ space can be divided into several non-overlapping polar shaped sectors. This division uses a center point and angular division. Center point $\boldsymbol{C}$ is calculated as the average of all corners of the inner initial polygon.

One way to divide the space is to use a uniform division of an angle from 0 to 2π. Using this, we have to calculate the exact angle between the vector $\boldsymbol{x}' = [0,1]^T$ and the vector $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{C}$, and such a calculation uses the following formula:

$$\varphi = \operatorname{arctg2}(\boldsymbol{v}_x, \boldsymbol{v}_y)\,. \tag{2}$$

Calculation of the function $\operatorname{arctg2}(\boldsymbol{v}_x, \boldsymbol{v}_y)$ takes a lot of time and we therefore use a simplified calculation of the approximated angle. The simplified angle is not uniformly distributed on a circle, but it is uniformly distributed on the border of a square $\langle -1, -1\rangle \times \langle 1,1\rangle$. When calculating the angle, we have to locate the exact half of quadrant, i.e. octant, where the point is located, and then calculate the intersection with the given side. The intersection with a side is simple, as all sides of the box's axis are aligned and intersect with the main axes at y or $x = 1$ or -1. The distribution of a simplified angle can be seen on Fig. 2.

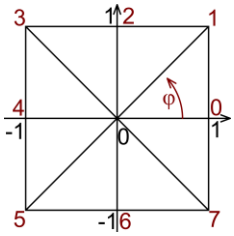


Fig. 2. Uniform distribution of a simplified angle on a unit square. Angle $\varphi \in \langle 0,8\rangle$ instead of $\langle 0,2\pi\rangle$.

Calculation of a simplified angle is faster than the formula (2) and gives almost the same results, as can be seen in Fig. 3.

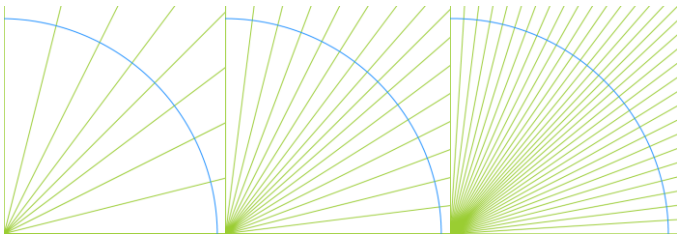


Fig. 3. Division of space into 32 (a), 64 (b) and 128 (c) non-overlapping sectors using uniform distribution of a simplified angle.

Now we have a simple calculation of the simplified angle and therefore we are able to determine the index of the sector to which the point belongs.

Each sector with the index i contains one maximum point $\boldsymbol{R}_i^{max}$. This point has the maximum (from all points in a sector) distance from the center point $\boldsymbol{C}$. The initial points $\boldsymbol{R}_i^{max}$ lies on the sides of the initial polygon, (see Fig. 4). We can calculate $\boldsymbol{R}_i^{max}$ as an intersection point of the middle axis of a sector and the polygon side.

All maximum points $\boldsymbol{R}_i^{max}$ form a polygon with vertices $\boldsymbol{R}_1^{max}$, …, $\boldsymbol{R}_K^{max}$, where K is count of all sectors (i.e. space division count).

For each new point we have to check whether the distance from the center $\boldsymbol{C}$ is greater than the distance for $\boldsymbol{R}_i^{max}$ from the center $\boldsymbol{C}$. If it is true, then we have to replace the maximum point $\boldsymbol{R}_i^{max}$ with the actual point, recalculate lines l^+ and l^-, (see Fig. 5) and add this point into the sector with index i. Otherwise, we have to continue with the next step.

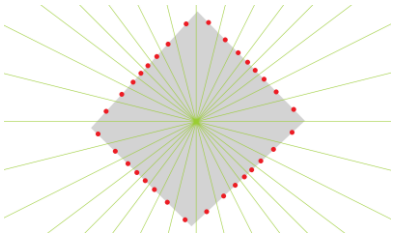


Fig. 4. Visualization of initial R_i^{max} points (red dots on the sides of the initial polygon).

The next step is to check whether the point lies over or under line segments l^+ and l^-, (see Fig. 5). We can compare the angle of the point with the angle of $\boldsymbol{R}_i^{max}$. If the angle is greater, then we have to use the line l^+, otherwise we have to use the line l^-. If the point is under the line l^+, or l^-, we can discarded it, because such a point cannot be part of the convex hull. Otherwise we have to add that point into the list of points associated with the segment with index i.

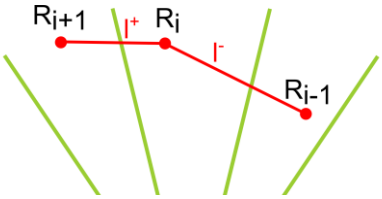


Fig. 5. Visualization of lines l^- and l^+.

C. Reduction of Suspicious Points

All suspicious points are already divided into sectors. In the beginning of the division process we set up points $\boldsymbol{R}_i^{max}$ to some initial values and used them to check whether it would be necessary to add or discard the tested point. The points $\boldsymbol{R}_i^{max}$ have changed during the division process, therefore we can recheck all the suspicious points using the final $\boldsymbol{R}_i^{max}$ points. This step will minimize the number of suspicious points that can be part of the convex hull. All the final suspicious points are connected with red straight lines, see Fig. 8.

D. Convex Hull Calculation

Convex hull computation can be done using any convex hull algorithm. The number of the input points compared to the number of the original points before reduction is extremely low, as can be seen Table 5. The time needed for convex hull computation is therefore not significant compared to the time required for the reduction of all the original input points. We can therefore use any algorithm for convex hull creation and not necessarily the fastest one. We chose to use the Graham Scan algorithm.

The input set of points for convex hull computation is subdivided into sectors. We can utilize the partial ordering of points in order to speed-up the convex hull algorithm. The first step of the Graham Scan is to sort all points in increasing order of the angle between vector $\boldsymbol{x}' = [0,1]^T$ and the vector $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{C}$, where the point $\boldsymbol{C}$ is the center point of the initial polygon and $\boldsymbol{x}$ is the point. All of these angles have already been precomputed from the point division phase and we do not have to compute them once more. The sorting process is done for each sector separately and then only the sorted groups of points are joined into one sorted array.

The Graham Scan needs one initial point which will be on the convex hull. We have to find the point with the highest x coordinate. This step takes $O(M)$, where M is the number of the suspicious points (the input points for this part of convex hull creation).

The algorithm proceeds by considering each of the points in the sorted array in sequence. For each point, it is determined whether moving from the two previously considered points to this point is a "left turn" or a "right turn":

$$[(\boldsymbol{B}-\boldsymbol{A})\times(\boldsymbol{C}-\boldsymbol{A})]_z \begin{cases} \geq 0 & \text{left turn} \\ < 0 & \text{right turn} \end{cases}, \tag{3}$$

where points $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are the three last points in that order and their z coordinate equals 0. The formula $[(\boldsymbol{B}-\boldsymbol{A})\times(\boldsymbol{C}-\boldsymbol{A})]_z$ means only the z coordinate from the cross product. If it is the "right turn", this means that the second point to the last point is not part of the convex hull and should be removed from consideration. This process continues as long as the set of the last three points is a "right turn". As soon as a "left turn" is encountered, the algorithm moves on to the next point in the sorted array. The algorithm ends when the last added and positively tested point is the starting point.

III. Experimental Results

The approach proposed has been implemented in C# using .Net Framework 4.5 and tested on data sets using PC with the configuration:

- CPU: Intel® Core™ i7 920 (4 × 2,67GHz),
- memory: 12 GB RAM,
- operating system Microsoft Windows 8 64bits

A. Distribution of Points

The proposed approach has been tested using several types of point distribution in $2D$. Some of the distributions were well known, randomly distributed uniform points inside a unit square and inside a unit circle. Another distribution used were points with Gaussian distribution and points lying on a unit circle. All of these distributions are well known. The last two distributions used are Halton points and Gauss Ring points. Both of these distributions are described in the following subchapters.

1. Halton Points

Halton sequence is a deterministic sequence of numbers that produces well-spaced "draws" from the unit interval. The sequence is based on a particular prime number and is constructed based on finer and finer prime-based divisions of sub-intervals of unit interval. An example of a Halton sequences based on prime numbers 2 and 3 start with the following numbers:

$$\begin{aligned} Halton(2) &= \frac{1}{2},\frac{1}{4},\frac{3}{4},\frac{1}{8},\frac{5}{8},\frac{3}{8},\frac{7}{8},\frac{1}{16},\frac{9}{16},\dots \\ Halton(3) &= \frac{1}{3},\frac{2}{3},\frac{1}{9},\frac{4}{9},\frac{7}{9},\frac{2}{9},\frac{5}{9},\frac{8}{9},\frac{1}{27},\dots \end{aligned} \tag{4}$$

When we pair the Halton sequences in (4) up, we get a sequence of points in $2D$ in a unit square:

$$\begin{aligned} Halton(2,3) = &\left(\frac{1}{2},\frac{1}{3}\right),\left(\frac{1}{4},\frac{2}{3}\right),\left(\frac{3}{4},\frac{1}{9}\right),\left(\frac{1}{8},\frac{4}{9}\right),\left(\frac{5}{8},\frac{7}{9}\right), \\ &\left(\frac{3}{8},\frac{2}{9}\right),\left(\frac{7}{8},\frac{5}{9}\right),\left(\frac{1}{16},\frac{8}{9}\right),\left(\frac{9}{16},\frac{1}{27}\right),\dots \end{aligned} \tag{5}$$

It can be seen that a Halton sequence covers the space more evenly than randomly generated uniform points, (see Fig. 6) (Halton, 1964).

Fig. 6. 10^3 $2D$ Halton points generated by $Halton(2,3)$ (left) and 10^3 $2D$ random points with uniform distribution (right).

2. Gauss Ring Points

This refers to a distribution of points in $2D$. The generation of such points is realized according to the following equation:

$$\begin{aligned} Point_{GaussRing} &= [\varphi, r] \\ &= [rand(0,2\pi), 1 + sign \\ &\quad \cdot random_{Gauss}] \end{aligned} \tag{6}$$

where $rand(0,2\pi)$ is a random number from $\langle 0, 2\pi \rangle$, $sign$ is randomly generated either number 1 or number (-1) and $random_{Gauss}$ is randomly generated number with Gauss distribution from interval $\langle 0, \infty \rangle$. A visualization of 10^3 Gauss Ring points can be seen in the Fig. 7.

Fig. 7. 10^3 $2D$ Gauss Ring points.

It can be seen that most of the points are relatively close to the unit circle and only a few points are far from that unit circle.

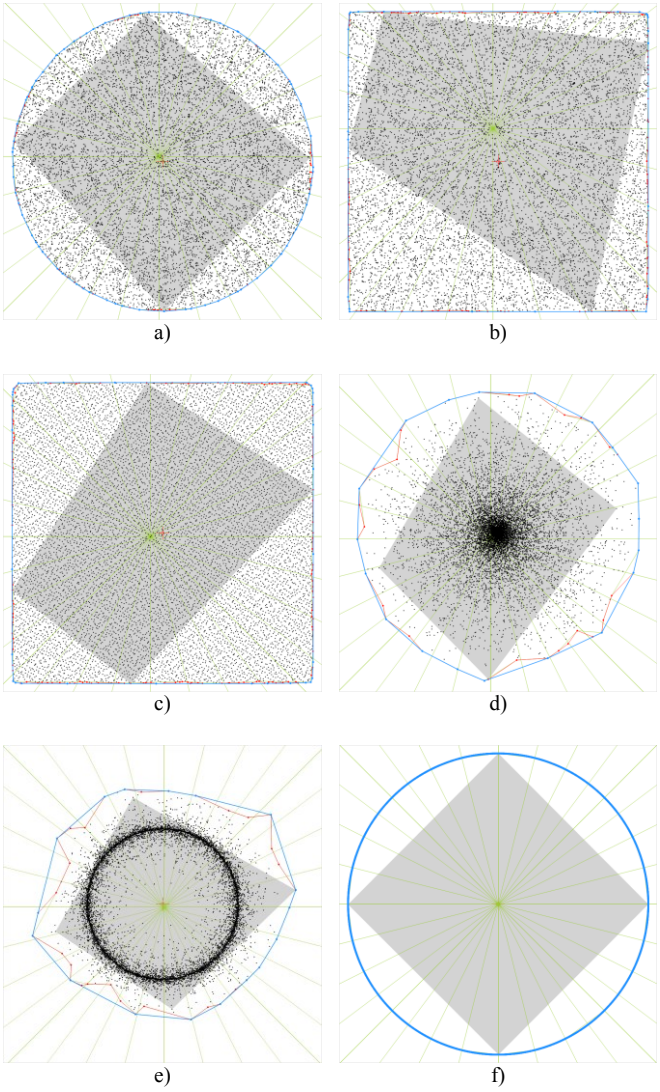


Fig. 8. Convex hulls of points with different distributions: a) uniform points in circle, b) uniform points in square, c) Halton points, d) Gauss points, e) Gauss Ring points and f) points on circle.

B. Examples of Convex Hull

We tested the algorithm proposed on datasets with different distribution of points. The results for 10^4 points are shown in Fig. 8. The grey polygon is the initial polygon, light green lines visualize the sectors of divided space, the red cross in the middle is the right center of all points, i.e. calculated from all input points, and the red straight lines are connected points that are the output of our reducing algorithm, i.e. the input for the convex hull calculation part.

C. Optimal Number of Divisions

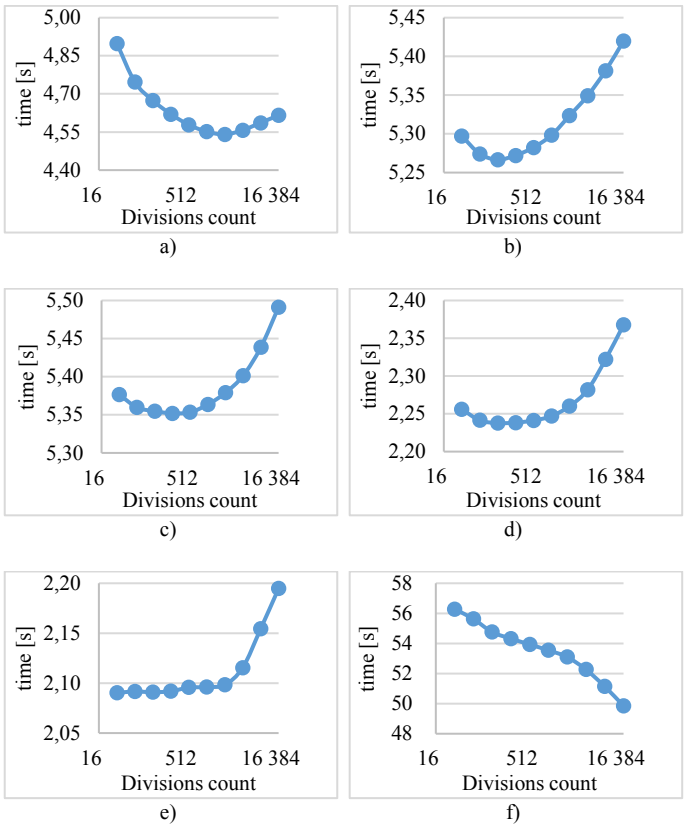
One of the first parts of the approach proposed is the division of input points into non-overlapping sectors. We need to know what the optimal number of divisions is. The optimal number of divisions should depend on the distribution of points, therefore we have to measure it for each type of distribution separately.

We measured the time complexity of the convex hull construction, when using our algorithm for reducing input points, for a different number of input points and for different types of point distributions and a different number of subdivisions. Examples of measured times for 10^8 points can be seen in Graph 1. The graph demonstrates that for uniform points in a circle or a square, as well as for Halton points, Gauss points and Gauss Ring points, the time complexity decreases with an increasing number of divisions. This happens up to an optimal number of sectors where the time complexity is minimal. From this number of divisions onwards, the time complexity increases with an increasing number of points per domain. A question which needs to be answered is: what is the optimal number of divisions for which the time performance is the best? For the last type of distribution, i.e. points on a circle, the situation is a bit different. As can be seen in Graph 1-f, the time complexity of convex hull decreases with a higher number of divisions. The reason for this is that all points, or most of them because of the float precision, are a part of the convex hull. Therefore a finer division means less points in each sector and a faster sorting procedure.

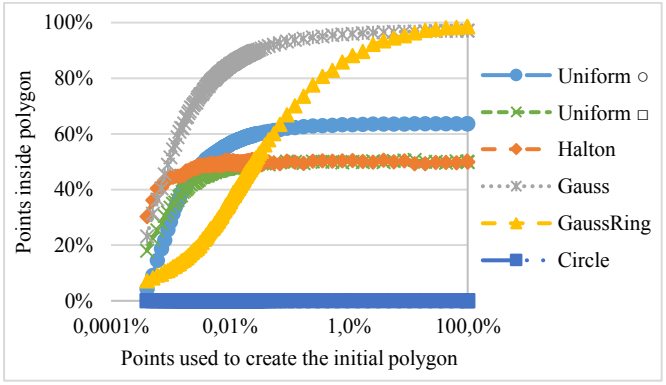
Using results from Graph 1 and other results for different number of input points, i.e. 10^6, $\sqrt{10}\cdot 10^6$, 10^7, $\sqrt{10}\cdot 10^7$ and 10^8, we end up with the following result. The optimal number of divisions is almost the same for all the number of input points.

D. Initial polygon creation

The initial polygon does not have to be created using all input points. We can randomly select only some points and find the initial polygon. We measured the number of points inside the initial polygon, when the polygon is created using some percentage of input points, (see Graph 2). It can be seen that when using more than 10% of the input points, the size of the initial polygon no longer changes.



Graph 1. The time performance of a convex hull algorithm, when using our algorithm for reducing input points, for different point distributions and different division counts. The number of input points is 10^8. Distribution of points are: a) uniform points in circle, b) uniform points in square, c) Halton points, d) Gauss points, e) Gauss Ring points and f) points on circle.



Graph 2. The percentage of points inside the initial polygon when using only some percentage of points to create the initial polygon.

E. Number of Points Processed at each Step

The main idea of the algorithm proposed is to remove as many points as possible before the convex hull construction calculation. The first step of the algorithm is to remove points inside the initial polygon. The percentage of points eliminated by the initial polygon can be seen in Table 2.

Table 2. The percentage of eliminated points by the initial polygon.

Number of points	Number of eliminated points [%]					
	Uniform ○	Uniform □	Halton	Gauss	Gauss Ring	Circle
1E+6	63,590	48,881	49,934	96,741	96,488	0,000
$\sqrt{10}$E+6	63,626	48,839	48,469	96,938	97,446	0,000
1E+7	63,644	50,070	48,941	97,093	98,795	0,000
$\sqrt{10}$E+7	63,653	49,967	49,693	97,153	99,128	0,000
1E+8	63,657	49,342	49,452	97,196	99,305	0,000

The second step of the algorithm is to remove points inside $\boldsymbol{R}^{max}$ polygon, see Table 3. The number of removed points is, together with Table 2, always higher than 99,6%, except of the points on a circle, where the number of eliminated points is 0.

Table 3. The percentage of eliminated points by the $\boldsymbol{R}^{max}$ polygon.

Number of points	Number of eliminated points [%]					
	Uniform ○	Uniform □	Halton	Gauss	Gauss Ring	Circle
1E+6	36,170	50,723	49,718	3,114	3,399	0,000
$\sqrt{10}$E+6	36,276	50,962	51,358	3,006	2,511	0,000
1E+7	36,310	49,825	50,969	2,886	1,190	0,000
$\sqrt{10}$E+7	36,320	49,976	50,258	2,840	0,866	0,000
1E+8	36,324	50,626	50,521	2,802	0,693	0,000

The next step is to recheck all suspicious points against the final $\boldsymbol{R}^{max}$ polygon. The percentage of points eliminated by this step is already small, as the number of suspicious points is low, as can be seen in Table 4.

Table 4. The percentage of eliminated points after recalculating all points using the final $\boldsymbol{R}^{max}$ polygon.

Number of points	Number of eliminated points [%]					
	Uniform ○	Uniform □	Halton	Gauss	Gauss Ring	Circle
1E+6	0,194	0,251	0,218	0,116	0,085	0,000
$\sqrt{10}$E+6	0,072	0,119	0,105	0,047	0,033	0,000
1E+7	0,028	0,059	0,052	0,018	0,013	0,000
$\sqrt{10}$E+7	0,012	0,031	0,026	0,007	0,005	0,000
1E+8	0,006	0,017	0,014	0,002	0,002	0,000

The number of input points for the final convex hull creation is really small, (see Table 5), because most of the points were discarded, as they cannot be the part of the convex hull. The only exception are points on a circle, because all of them are the input for convex hull.

Table 5. The percentage of points as the input for convex hull.

Number of points	Number of input points [%]					
	Uniform ○	Uniform □	Halton	Gauss	Gauss Ring	Circle
1E+6	0,046	0,145	0,129	0,029	0,028	100,000
$\sqrt{10}$E+6	0,026	0,080	0,068	0,009	0,009	100,000
1E+7	0,019	0,045	0,039	0,003	0,003	100,000
$\sqrt{10}$E+7	0,015	0,026	0,023	0,001	0,001	100,000
1E+8	0,014	0,015	0,013	0,000	0,000	100,000

The number of points on the convex hull can be seen in Table **6**. Comparing this table to the Table 5, we can see that at least one third of points from Table 5 are on the convex hull. This is true for all distributions except for points on a circle. The number of points lying on the convex hull for points on a circle is less than 50%. The reason for this is the calculation which uses float precision and resulting in the impossibility of the generated points cannot lying on the exact circle with radius one; but it is a very untypical case anyway.

Table 6. The percentage of points on the convex hull.

Number of points	Number of points [%]					
	Uniform ○	Uniform □	Halton	Gauss	Gauss Ring	Circle
1E+6	0,034	0,005	0,004	0,006	0,004	46,646
$\sqrt{10}$E+6	0,016	0,004	0,001	0,002	0,001	46,359
1E+7	0,008	0,007	0,002	0,001	0,000	45,788
$\sqrt{10}$E+7	0,006	0,008	0,006	0,000	0,000	44,834
1E+8	0,005	0,006	0,005	0,000	0,000	43,836

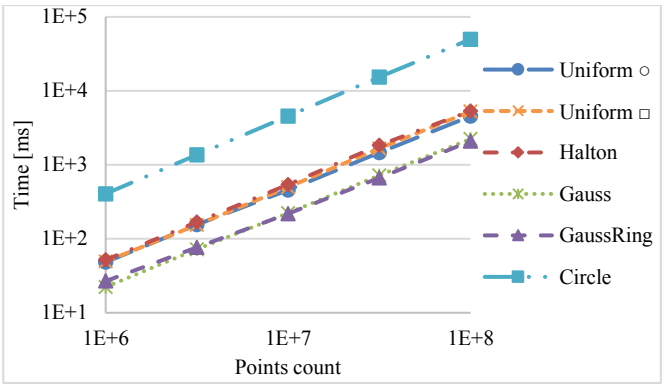
F. Time Performance

In some applications, time performance is one of the most important criteria. We measured running times for the convex hull, for different numbers of input points and for different distributions of points, when using our algorithm for reducing input points. The running times were measured many times and the average times for each number of points and each type of distribution are presented in Table 7.

Table 7. The time performance of convex hull for different numbers of input points and different distributions of points, when using our algorithm for reducing input points. Time is in milliseconds [ms].

Number of points	Time [ms]					
	Uniform ○	Uniform □	Halton	Gauss	Gauss Ring	Circle
1E+6	48,3	49,3	51,8	22,3	26,8	406,1
$\sqrt{10}$E+6	155,2	154,3	167,6	71,9	76,6	1 357,4
1E+7	457,5	496,4	538,3	222,7	218,9	4 547,1
$\sqrt{10}$E+7	1 460,6	1 662,7	1 843,2	719,2	662,5	15 272,2
1E+8	4 540,7	5 266,7	5 352,0	2 237,7	2 090,2	49 851,7

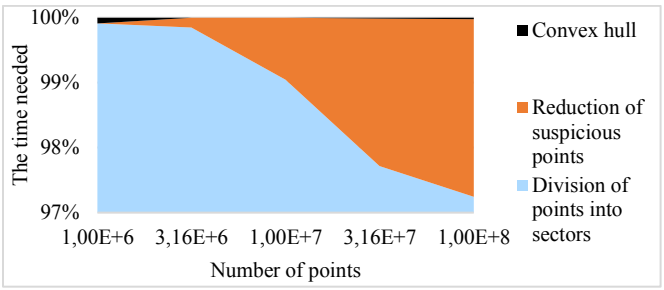
It can be seen that the fastest running times are for the Gauss Ring and Gauss distributions. This is because most of the points in Gauss distribution lie inside the initial polygon and moreover, the number of points on the convex hull is very small for the Gauss Ring and Gauss distribution. The worse time performance is for points on a circle. Because of the float precision, all points, or almost all of them, are on the convex hull and no points can be discarded. The running times for uniform distribution inside a square and for a Halton distribution are almost the same. The times for points with uniform distribution inside a circle are a bit faster than the times for points with uniform distribution inside a square, because the number of points inside the initial polygon is higher for uniform points inside a circle than inside a square. The times from Table 7 can be seen for better comparison in Graph 3.



Graph 3. The time performance of convex hull, using our speed-up algorithm, for different number of input points and different distribution of these points (note that both axes have a logarithmical scaling).

Convex hull algorithm consists of several steps. We can measure these steps and see the percentage proportion between them. The first measured step is to divide the points into segments and discard some of them. The second step is to reduce the suspicious points and the last step is the calculation of the convex hull from the selected points. The comparison of the perceptual time performance of these three steps can be seen in Graph 4.

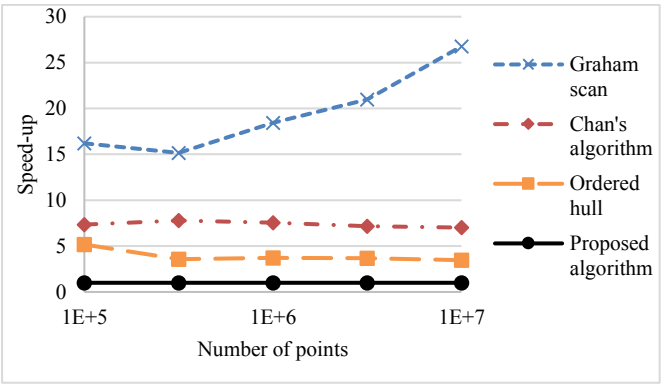
It can be seen that the most time consuming part of the approach proposed is the step in which the points are divided into sectors (lower part in a graph). This step takes from almost 100% to at least 97% of the total time of the convex hull creation. The second most time consuming part is the reduction of suspicious points, which takes up to 3% of the total time. The step of convex hull calculation itself takes almost no time, i.e. $\ll$ **1% of the total time** (the top black part of the diagram) compared to the total time of the proposed algorithm.



Graph 4. The time needed on average for each step of the convex hull. Note, that the vertical axis values are from 97% to 100%.

G. *Comparison with Other Algorithms*

There are many algorithms for convex hull calculation. We compared our reducing algorithm using Graham Scan with three more known algorithm. One algorithm, i.e. the Graham Scan, is of the $O(n \log n)$ time complexity, where n is the number of input points. The two other algorithm, i.e., Chan's algorithm and Ordered hull, are of the $O(n \log h)$ time complexity, where h is the number of points lying on the convex hull. The results can be seen in Graph 5 and were measured for a different number of points with uniform distribution inside a square. It should be noted that all algorithms are implemented using the same programming language, i.e. in C# using .Net Framework 4.5.



Graph 5. The speed-up of Graham Scan algorithm using our reducing algorithm compared to other convex hull algorithms for points in square with uniform distribution.

According to the results from Graph 5 our algorithm is on average:

- 3,6 times faster than Ordered hull,
- 7,4 times faster than Chan's algorithm,
- more than 20 times faster than Graham Scan.

The proposed convex hull algorithm was tested on real datasets as well. Two datasets were derived from $3D$ mesh models by projecting the vertices of each $3D$ model onto the

XY plane. These mesh models presented in Fig. 9 are directly obtained from the Stanford $3D$ Scanning Repository[1].

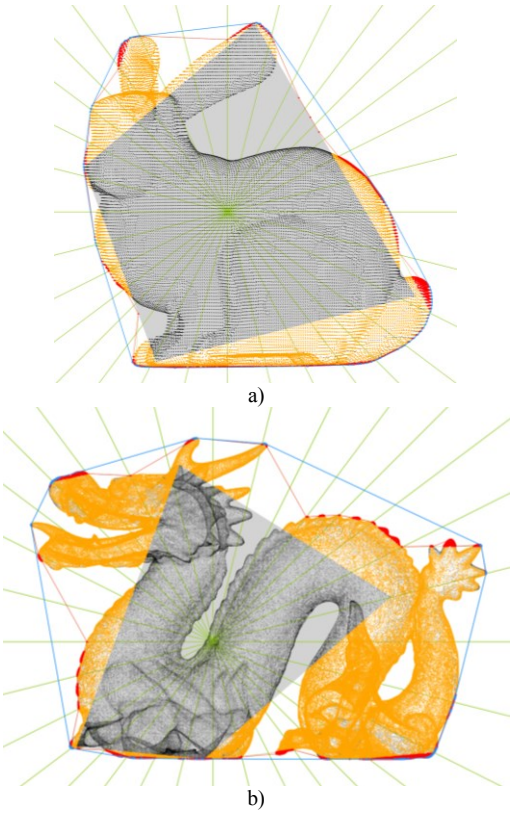
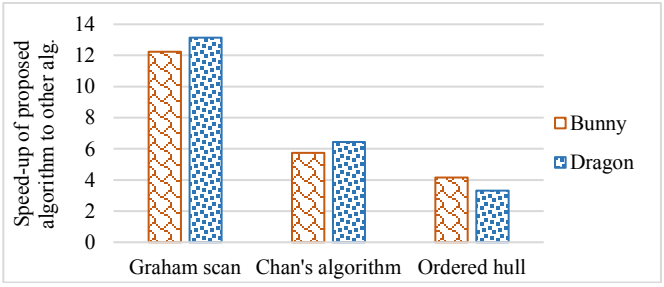


Fig. 9. Convex hull of two real datasets. The bunny contains 35 947 points (a) and the dragon contains 437 645 points (b).

The proposed algorithm was compared with other $2D$ convex hull algorithms and the speed-up of our algorithm can be seen in Graph 6. It can be seen, that the speed-up is similar to the speed-up of convex hull on points with uniform distribution inside a square.



Graph 6. The speed-up of the proposed reducing algorithm compared to other convex hull algorithms for real points datasets.

[1] http://www.graphics.stanford.edu/data/3Dscanrep/

IV. Conclusion

A new fast and easy way to implement algorithm for reducing points for convex hull calculation in E^2 has been presented. It uses the polar space division technique to speed up computation. The proposed algorithm proved robustness for different point distributions. The algorithm proposed is convenient for large data sets.

In the future, the algorithm will be modified to enable the parallel processing as many steps are independent and can easily be parallelized. The algorithm was be extended to $3D$, see (Skala et al, 2016).

Acknowledgments.

The authors would like to thank their colleagues at the University of West Bohemia, Plzen, for their comments and suggestions, and anonymous reviewers for their valuable comments and hints provided. The research was supported by MSMT CR projects LH12181 and SGS 2016-013.

References

Andrew, Alex M.: Another efficient algorithm for convex hulls in two dimensions, Inf. Processing Letters, Vol.9, No.5, pp.216-219, 1979

Bykat, A.: Convex hull of a finite set of points in two dimensions, Information Processing Letters, Vol.7, No.6, pp.296-298, 1978

Eddy, William F.: A new convex hull algorithm for planar sets, ACM Transactions on Mathematical Software, Vol.3, No.4, pp.398-403, 1977

Graham, Ronald L.: An efficient algorithm for determining the convex hull of a finite planar set, Inf. processing letters, Vol.1, No.4, pp.132-133, 1972

Halton, J.: Algorithm 247: Radical-inverse quasi-random point sequence. ACM, 1964, Vol. 7, No. 12, pp. 701-702

Chan, Timothy M.: Optimal output-sensitive convex hull algorithms in two and three dimensions, Discrete & Computational Geometry, Vol.16, No.4, pp.361-368, 1996

Chand, Donald R. and Kapur, Sham S.: An Algorithm for Convex Polytopes, Journal of the ACM, Vol.17, No.1, pp.78-86, http://dx.doi.org/10.1145/321556.321564, 1970

Jarvis, Ray A.: On the identification of the convex hull of a finite set of points in the plane, Information Processing Letters, Vol.2, No.1, pp.18-21, 1973

Kallay, M.: The complexity of incremental convex hull algorithms in R d, Information Processing Letters, Vol.19, No.4, pp.197, 1984

Kirkpatrick, David G. and Seidel, R.: The ultimate planar convex hull algorithm?, SIAM journal on computing, Vol.15, No.1, pp.287-299, 1986

Liu, G. and Chen, Ch.: A new algorithm for computing the convex hull of a planar point set, Journal of Zhejiang University SCIENCE A, Vol.8, No.8, pp.1210-1217, 2007

Preparata, Franco P. and Hong, Se J.: Convex hulls of finite sets of points in two and three dimensions, CACM, Vol.20, No.2, pp.87-93, 1977

Skala,V.: Fast $O_{expected}(N)$ Algorithm for Finding Exact Maximum Distance in E2 Instead of O(N2) or O(N logN), ICNAAM 2013, AIP Conf.Proceedings No.1558, pp.2496-2499, AIP Publishing, 2013

Skala,V., Majdisova,Z.: Fast Algorithm for Finding Maximum Distance with Space Subdivision in E2, ICIG 2015 proceedings Part II, LNCS 9218, China, pp.261-274, ISSN 0302-9743, Springer, 2015

Skala,V., Majdisova,Z., Smolik,M.: Space Subdivision to Speed-up Convex Hull Construction in E3, Advances in Software Engineering, Vol.91, pp.12-22, Elsevier, ISSN 0965-9978, 2016

Skala,V., Smolik,M.: A Point in Non-Convex Polygon Location Problem Using the Polar Space Subdivision in E2, ICIG 2015 proceedings Part I, LNCS 9217, China, pp.394-404, ISSN 0302-9743, Springer, 2015