PAPER Special Section on VLSI Design and CAD Algorithms

A Hardware Oriented Approximate Convex Hull Algorithm and its FPGA Implementation*

Tatsuma MORI[†a)], **Taito MANABE**[†], *Nonmembers*, *and* **Yuichiro SHIBATA**[†], *Member*

SUMMARY The convex hull is the minimum convex surrounding a given set of points. Since the process of finding convex hulls has various practical application fields including embedded real-time systems, efficient acceleration of convex hull algorithms is an important problem in computer geometry. In this paper, we discuss an FPGA acceleration approach to address this problem. In order to compute the convex hull of an unsorted point set, it is necessary to store all the points during the computation, and thus the capacity of a on-chip memory is likely to be a major constraint for efficient FPGA implementation. On the other hand, approximate convex hulls are often sufficient for practical applications. Therefore, we propose a hardware oriented approximate convex hull algorithm, which can process the input points as a stream without storing all the points in the memory. We also propose some computation reduction techniques for efficient FPGA implementation. Then, we present FPGA implementation of the proposed algorithm, which is parallelized both in temporal and spatial domains, and evaluate its effectiveness in terms of performance and accuracy. As a result, we demonstrated 11 to 30 times faster performance compared to the widely-used convex hull software library Qhull. In addition, accuracy assessment revealed that the maximum approximation error normalized to the diameters of point sets was 0.038%, which was reasonably small for practical use cases.

key words: *convex hull, hardware algorithm, FPGA, approximated algorithm*

1. Introduction

Given a point set P, the minimum convex that includes all the points in P is called the convex hull of P. For example,for point sets defined on a two dimensional (2D) planar space, the convex hull of P becomes the convex polygon whose vertices are a subset of P as illustrated in Fig. 1. A convex hull problem, whose purpose is to find the convex hull $CH(P)$ of a given point set P, has various practical applications [2], and thus its high speed algorithms and implementation have been widely addressed in the field of computational geometry.

Qhull [3] is one of the most popular software libraries for computing convex hulls. The main algorithm used in the Qhull library is called Quickhull [4], [5]. Mei proposed a GPU-accelerated high-speed 2D convex hull algorithm called CudaChain [6]. CudaChain firstly reduces interior points of a given point set to form a simple polygon. This step is parallelly performed on a GPU. Then, the convex hull of the simplified polygon is computed with Melkman's algorithm [7] on a CPU. Performance evaluation revealed that CudaChain with GT640 and GTX660M GPUs outperforms Qhull by a factor of 5 to 6 for a 20M-point set. Srungarapu et al. implemented the Quickhull algorithm on a GTX280 GPU and achieved 14 times speedup to Qhull [8]. Srikanth et al. also parallelized the Quickhull algorithm and evaluated the performance on a GTX280 GPU as well as a Cell Broadband Engine [9]. The GPU implementation achieved approximately 15 times speedup to Quickhull implementation on a CPU, while Cell Broadband Engine implementation obtained 4 to 5 times performance compared to the main processor execution. Tzeng et al. implemented a general GPU framework for divide-and-conquer algorithms and applied it to accelerate 2D and 3D convex hull computations [10]. Compared to Qhull implementation, they demonstrated more than 10 times acceleration.

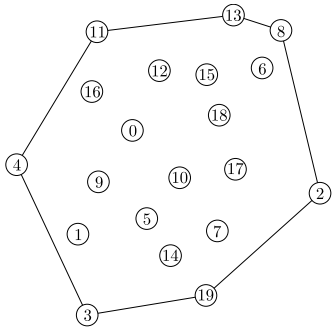


Fig. 1 Example of convex hull for a 2D point set.

Aiming at effective real-time convex hull calculation, Kemmotsu et al. proposed an FPGA-based parallel approach for convex hull computing for points in 2D images [11]. The basis of this approach is Andrew's Monotone Chain [12], which needs the input points to be sorted in advance as a pre-processing step. Kemmotsu's approach eliminates the pre-processing step from the implementation by limiting application domains to image processing, in which input points are always given in a raster scan manner. They also proposed to parallelly compute the upper hull and the lower hull. The final result is obtained by combining them. However, the approach is not suitable for general systems that need to handle unsorted input points.

In typical convex hull algorithms for unsorted point sets, such as Graham's scan [13], Andrew's Monotone Chain [12], and Quickhull [4], entire input points need to be stored and

Manuscript received March 15, 2021.
Manuscript revised July 13, 2021.
Manuscript publicized September 2, 2021.

[†]The authors are with Graduate School of Engineering, Nagasaki University, Nagasaki-shi, 852-8521 Japan.

*This paper was presented at the Eighth International Symposium on Computing and Networking (CANDAR), Online, Nov. 24–27, 2020 [1].

a) E-mail: tatsuma@pca.cis.nagasaki-u.ac.jp

DOI: 10.1587/transfun.2021VLP0016

available in memory before starting computation, which imposes a challenge for FPGA implementation. Since the capacity of on-chip memory of FPGAs is limited, the use of external memory is required to avoid size (the number of points) limitation of solvable problems. However, this approach will significantly reduce the advantage of FPGAs, in terms of both performance and power efficiency. For FPGAs, online algorithms that can process the input in a point-by-point manner are more preferable.

Fortunately, convex hulls are often used to roughly grab abstract object shapes in applications such as collision detection and path planning, where computation of the exact convex hulls is not necessarily needed. For such application domains, approximate convex hull computing algorithms [14]–[17] are acceptable. In this paper, we propose an online approximate convex hull algorithm for unsorted point sets and present its FPGA implementation. This algorithm is hardware oriented, in a sense that input points can be processed in a pipelined manner without storing all the points in memory. As far as the authors' knowledge, FPGA implementation of such online approximate convex hull algorithms has not been reported so far.

The rest of the paper is organized as follows. The proposed algorithm is described in Sect. 2, and the FPGA implementation is shown in Sect. 3. In Sect. 4, we evaluate the implementation by comparing the performance to related work [6] and the software library Qhull [3]. Finally, Sect. 5 concludes this paper.

2. Algorithm

Let θ be an angle between a unit vector $\boldsymbol{v} \in \mathbf{R}^2$ and a position vector $\boldsymbol{P} \in \mathbf{R}^2$. Let us consider an inner product map $d : \mathbf{R}^2 \to \mathbf{R}$ such that:

$$d(\boldsymbol{P}) \equiv \boldsymbol{v}^T \boldsymbol{P} = \|\boldsymbol{v}\|\|\boldsymbol{P}\| \cos\theta = \|\boldsymbol{P}\| \cos\theta \tag{1}$$

where θ is the angle between the two vectors P and v. Let ℓ be the line with the direction vector $\boldsymbol{v}$ passing through the origin. As depicted in Fig. 2, $d(\boldsymbol{P})$ can be interpreted as the signed distance between the origin and the point $\boldsymbol{P}'$, which is the projection of point $\boldsymbol{P}$ on the line ℓ.

Let us consider that all the points in a given point set P are projected onto a line ℓ. If a projected point becomes one of the end points on the line ℓ, the corresponding original point is one of the vertices of the convex hull $CH(P)$ as shown in Fig. 3 and Fig. 4. Therefore, the vertices of $CH(P)$ can be obtained by finding the points $\boldsymbol{P} \in P$ which have the maximum or minimum value of $d(\boldsymbol{P})$, by rotating the unit vector $\boldsymbol{v}$. Here, we call $\boldsymbol{v}$ a scan vector. Although ideally the scan vector should be continuously rotated to scan the 2D plane, we need to discretize the algorithm using a small step angle for implementation. By rotating the scan vector in the counterclockwise direction, the vertices of the convex hull can be obtained in a counterclockwise order as illustrated in Fig. 5.

While this principle is also shared by other approximated convex hull algorithms such as [14] and [17], we introduce a computation reduction technique for FPGA implementation as follows. A naive calculation of one inner product in Eq. (1) requires two multiplications. Utilizing symmetry, however, four inner products can be computed by two multiplications. With a polar coordinates system, a unit vector $\boldsymbol{v}$ is defined with a polar angle θ as:

$$\boldsymbol{v}(\theta) = (v_x, v_y)^T = (\cos\theta, \sin\theta)^T. \tag{2}$$

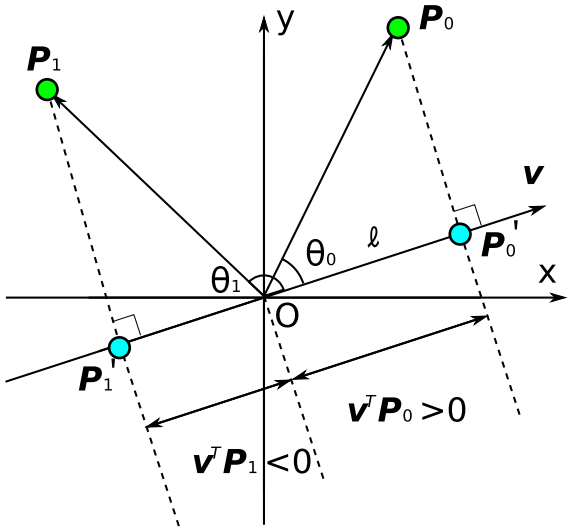


Fig. 2 Projection points and signed distances.

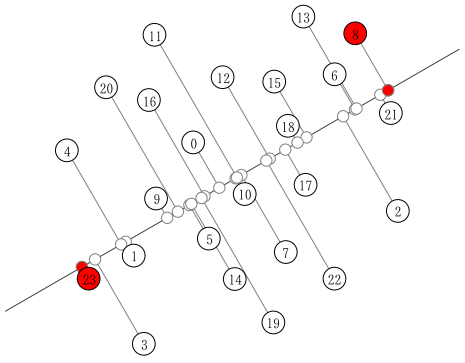


Fig. 3 Projection of points to a line. Nodes 8 and 23 are the end points.

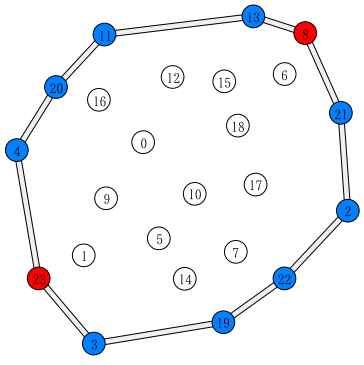


Fig. 4 Convex hull of the points shown in Fig. 3. Node 8 and 23, which are the end points in the projection, belong to the vertices of the convex hull.

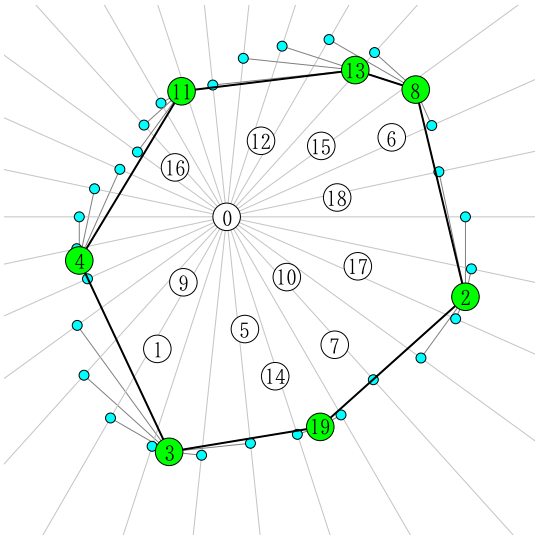


Fig. 5 Lines made by rotating scan vectors and their projected end points (light blue nodes). Their original points form the vertices of the convex hull (green nodes).

Here, let us consider the following four symmetric vectors:

$$\begin{aligned} &\boldsymbol{v_0} = \boldsymbol{v}(\theta),\ \ \boldsymbol{v_1} = \boldsymbol{v}\left(\frac{\pi}{2} - \theta\right),\ \ \boldsymbol{v_2} = \boldsymbol{v}\left(\theta + \frac{\pi}{2}\right), \\ &\boldsymbol{v_3} = \boldsymbol{v}\left(\pi - \theta\right) \end{aligned} \tag{3}$$

where $\theta \in [0, \frac{\pi}{4})$. The inner products between point $\boldsymbol{P} = (p_x, p_y)^T$ and these vectors can be expressed as:

$$\begin{aligned} d_0(\boldsymbol{P}) &\equiv \boldsymbol{v_0}^T\boldsymbol{P} = p_x\cos\theta + p_y\sin\theta \\ d_1(\boldsymbol{P}) &\equiv \boldsymbol{v_1}^T\boldsymbol{P} = p_x\sin\theta + p_y\cos\theta \\ d_2(\boldsymbol{P}) &\equiv \boldsymbol{v_2}^T\boldsymbol{P} = -p_x\sin\theta + p_y\cos\theta \\ d_3(\boldsymbol{P}) &\equiv \boldsymbol{v_3}^T\boldsymbol{P} = -p_x\cos\theta + p_y\sin\theta. \end{aligned} \tag{4}$$

Dividing the both sides of Eq. (4) by $\cos\theta$, we get:

$$\begin{aligned} d'_0(\boldsymbol{P}) &= p_x + p_y\tan\theta \\ d'_1(\boldsymbol{P}) &= p_x\tan\theta + p_y \\ d'_2(\boldsymbol{P}) &= -p_x\tan\theta + p_y \\ d'_3(\boldsymbol{P}) &= -p_x + p_y\tan\theta \end{aligned} \tag{5}$$

where $d'_i(\boldsymbol{P}) \equiv \frac{d_i(\boldsymbol{P})}{\cos\theta}$. Since $\cos\theta > 0$ for $\theta \in [0, \frac{\pi}{4})$,

$$d_i(\boldsymbol{P_0}) \le d_i(\boldsymbol{P_1}) \Leftrightarrow d'_i(\boldsymbol{P_0}) \le d'_i(\boldsymbol{P_1}) \tag{6}$$

for any pair of points $\boldsymbol{P_0}$ and $\boldsymbol{P_1}$. Therefore, we can use $d'_i(\boldsymbol{P})$ in stead of $d_i(\boldsymbol{P})$ for judgment of end points. By pre-calculating the value of $\tan\theta$, the four inner products in Eq. (5) can be computed with two multiplications, that is, $p_x\tan\theta$ and $p_y\tan\theta$.

The proposed approximate convex hull algorithm is described in Algorithm 1. Here, n and s are the size of a given point set and the number of scan vectors in $[0, \frac{\pi}{4})$, respectively. In this algorithm, a circle is divided into eight sectors as shown in Fig. 6, and the point numbers (IDs) on the convex hull are pushed into each stack $N_0, N_1, \cdots, N_7$ in a scanned order. Since the computation for one scan takes $O(n)$, the time complexity for sequential execution of this algorithm is $O(ns)$. If we make s a small value, the execution time will be reduced, but the approximation error will increase.

Algorithm 1 Approximate convex hull algorithm

Input: $P = \{\boldsymbol{P_0}, \boldsymbol{P_1}, \cdots, \boldsymbol{P_{n-1}}\}$: Point set
s : number of scan vectors in $[0, \frac{\pi}{4})$
Output: Array of point IDs for convex hull vertices in counterclockwise
Data Structure: $N_0, N_1, \cdots N_7$: Stacks

```
for i = 1, ··· , n − 1 do
    P_i ⇐ P_i − P_0;
end for
for i = 0, ··· ,s −1 do
    θ ⇐ iπ/4s;
    for k = 0, 1, 2, 3 do
        n_k = arg max_{0≤j<n} d'_k(P_j); n_{k+4} = arg min_{0≤j<n} d'_k(P_j);
    end for
    for k = 0, ··· , 7 do
        if i = 0 or n_k ≠ N_k.top() then
            N_k.push(n_k);
        end if
    end for
end for
for k = 0, ··· , 3 do
    N_{2k+1}.reverse();
end for
return unite(N_0, ··· , N_7)
```

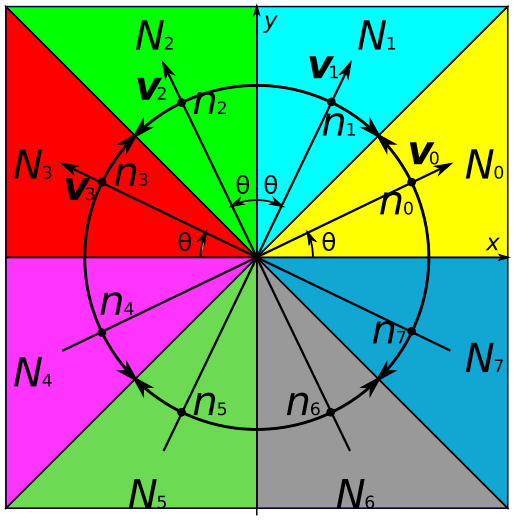


Fig. 6 Domains of angles for each stack.

3. Implementation

We designed pipeline-based custom hardware for the proposed approximate convex hull algorithm. Since the calculation for each scan vector is independent, we prepared as many calculation modules as the number of scan vectors to extract parallelism. The overall diagram of the circuit is as shown in Fig. 7. The roles of each signal and parameter are summarized in Table 1. The implemented circuit receives information for one point (point ID and xy coordinates) ev-

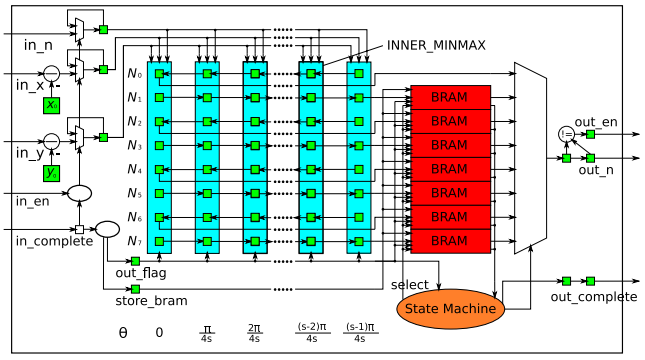


Fig. 7 The overall diagram of the circuit.

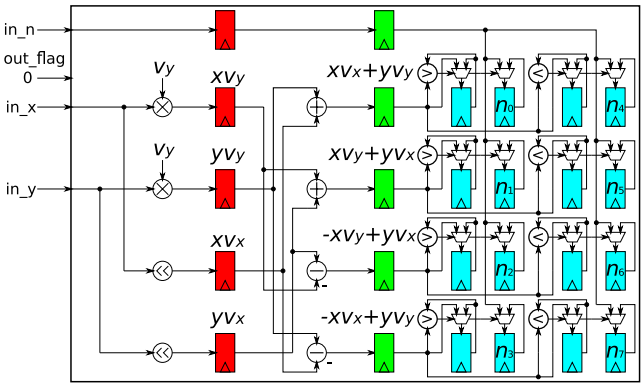


Fig. 8 INNER_MINMAX circuit (out_flag=0).

Table 1 The roles of each signal and parameter.

	name	role
input	in_n	point ID of input point
	in_x	x coordinate of input point
	in_y	y coordinate of input point
	in_en	input enable flag
	in_complete	input complete flag
output	out_n	point ID of CH vertex
	out_en	output enable flag
	out_complete	output complete flag
parameter	s	number of scan vectors in $[0, \frac{\pi}{4})$
	W_s	multiplication max bit width
	W_n	point ID bit width
	W_x	input x coordinate bit width
	W_y	input y coordinate bit width

ery clock cycle. The circuit starts output of the point IDs for the vertices of the calculated convex hull one by one each cycle, several clock cycles after whole point information has been given.

The horizontally arranged rectangles in Fig. 7 show INNER_MINMAX circuits, for each of which a unique scan vector $\boldsymbol{v}$ is assigned. This circuit calculates the inner products between the input point and the four symmetric vectors $\boldsymbol{v_0}, \cdots, \boldsymbol{v_3}$, and holds the maximum and minimum values as well as the corresponding point IDs. After the input of the point set and the calculation of the INNER_MINMAX circuit are completed, the approximate convex hull is obtained by outputting the point IDs in a counterclockwise order while eliminating duplication.

The xy coordinate values of the first input point P_0 are stored in x_0 and y_0 registers depicted in Fig. 7. The coordinates of the subsequent input points are transformed so that P_0 is the origin by subtracting x_0 and y_0. Each INNER_MINMAX circuit reads n, x, y and updates the provisional maximum and minimum points by calculating the inner products. The signal "in_en" is an input enable signal, thus the register values are not updated when this port is deasserted. The signal "in_complete" represents the completion of the input stream. Once this port is asserted, the subsequent input signals are just ignored without being stored in the registers.

The update of the INNER_MINMAX circuit is finished several clock cycles after "in_complete" becomes 1, and output of the result starts. After output starts, "out_flag" goes to 1. Meanwhile, another flag "store_bram" becomes 1 for $(s+1)$ clock cycles. Then the point IDs in the region N_0 ($\theta \in [0, \frac{\pi}{4})$) are outputted from "out_n" in the counterclockwise order, while the point IDs in $N_1 \sim N_7$ are stored in BRAM, eliminating duplicates. Note that the point IDs in N_1, N_3, N_5, and N_7 are stored from the $\theta = \frac{\pi}{4}$ side, so that they are obtained in the counterclockwise order.

The output of the IDs from N_0 is completed as soon as "store_bram" becomes 0, and the point IDs stored in BRAM are outputted from "out_n" in the order of $N_1 \sim N_7$ thereafter. Two clock cycles after the completion of N_7 output, "out_complete" goes to 1 to show the completion of the output.

For each INNER_MINMAX circuit,

$$\boldsymbol{v} = (v_x, v_y)^T = \left(2^{W_s}, \lfloor 2^{W_s} \tan\theta \rfloor\right)^T \tag{7}$$

is assigned as a scan vector. Since the norm of $\boldsymbol{v}$ can be changed to any value provided that the orientation is kept, both v_x and v_y are right-shifted so that the LSB of v_y becomes 1, aiming at hardware resource reduction. When "out_flag" is 0, the circuit receives input points from "in_n", "in_x", and "in_y", and updates the maximum and minimum values of the inner products and the point IDs as Fig. 8 shows. When "out_flag" is 1, the point IDs given from the outside are stored in the registers $(n_0, \cdots, n_7)$, which are used to store point IDs in Fig. 8, in order to form the shift register shown in Fig. 7.

In this architecture, it takes $(n+4)$ clock cycles to process all the input points with INNER_MINMAX circuits. Since the maximum number of output points is $8s$, which corresponds to the situation there is no duplication in the detected convex hull vertices, at most it takes $(8s+2)$ clock cycles to output the results. Therefore, the theoretical execution time T can be expressed as

$$T \leq \frac{n + 8s + 6}{F_{\max}} \tag{8}$$

where $F_{\max}$ is the maximum clock frequency of the system. Thus, the time complexity of the FPGA execution is $O(n+s)$, in contrast to the time complexity for sequential execution of

$O(ns)$.

Unlike to other convex hull algorithms, the proposed method does not adopt preprocessing to decimate input points. In general, the time complexity of the exact (non-approximate) convex hull algorithms is $O(n \log n)$. Thus, it is reasonable to perform preprocessing in $O(n)$ to reduce the number of points to be processed. On the other hand, the proposed approach calculates an approximated convex hull in $O(n + s)$. Therefore, decimation of input points is not considered for this approach.

4. Evaluation

The designed hardware was evaluated targeting on a Xilinx Virtex UltraScale FPGA (xcvu095-ffva2104-2-e-es2). For logic synthesis and mapping, Xilinx Vivado 2018.3 was used. For simulation, Cadence Xcelium 18.9 was used. For evaluation of execution time and approximation errors, the benchmark point sets used in the related work [6] were used. The benchmark sets consist of three groups: rbox, circle, and model. The rbox group contains point sets generated using Qhull's rbox, and their sizes (the number of points) are 0.1M, 0.2M, 0.5M, 1M, 2M, 5M, 10M, and 20M. The circle group contains point sets randomly generated within the unit circle, and their sizes are the same as those for the rbox group. The model group contains point sets obtained by projecting the vertices of 3D models onto the XY plane. The 3D models and their sizes are shown in Table 2. These models were obtained from the Stanford 3D Scan Repository [18] and the GIT Large Geometry Models Archive [19].

For the circuit parameters shown in Table 1, we evaluated four types of $s = 50, 150, 250$, and 350 ($W_s = \lceil \log_2 s \rceil$). W_n is set to 25 bits so that the maximum size of evaluation point sets (20M) can be addressed. Both W_x and W_y are set to 32 bits. At each rising clock edge, one point ID and corresponding real type xy coordinate values are read from the input point set. The coordinate values are multiplied by 2^{20} and the integer values are given to the circuit by truncating the fractional part.

For comparison, we evaluated the performance of software execution using the Qhull library (2019) on a PC with the following environments:

- CPU: Intel i9-9900K (3.6 GHz), Memory: 16 GB, OS: CentOS Linux 7.6.1810

We also compare the performance with GPU implementation described in [6], where the following two computational environments were used:

- GPU: GT640, CPU: Intel i5-3470, Memory: 8 GB, OS: Window 7 Pro
- GPU: GTX660M, CPU: Intel i7-3610QM, Memory: 6 GB, OS: Window 7 Pro

Table 2 Models and size.

Model	Size	Model	Size
Armadillo	0.17M	Turbine blade	0.88M
Angel	0.24M	Vellum manuscript	2.16M
Skeleton hand	0.33M	Asian Dragon	3.61M
Dragon	0.44M	Thai statue	5.00M
Happy Buddha	0.54M	Lucy	14.03M

For the execution time of Qhull, the average of 18 runs was used. The execution time for the FPGA implementation was obtained by dividing the number of clock cycles from the start of input to the end of output in HDL simulation with the maximum operating frequency obtained by Vivado STA.

To evaluate accuracy of the proposed approximate convex hull algorithm, we utilized several metrics as follows. Let $CH(P)$ be the convex hull obtained by Qhull for the point set P, and $CH'(P)$ be the approximate convex hull obtained by this implementation. Let $V = \{\boldsymbol{v_0}, \boldsymbol{v_1}, \cdots, \boldsymbol{v_{h-1}}\}$ and $V' = \{\boldsymbol{v'_0}, \boldsymbol{v'_1}, \cdots, \boldsymbol{v'_{h'-1}}\}$ be the vertices of the convex hull $CH(P)$ and $CH'(P)$, respectively. The order of both vertex sets is counterclockwise. The boundaries $\partial CH(P)$ of $CH(P)$ and $\partial CH'(P)$ of $CH'(P)$ are polygons that connect V an V', respectively. We consider an approximation evaluation metric r, which is defined as the ratio of the number of points that belong to the vertices of the approximate convex hull to the vertices of the exact convex hull, that is,

$$r = \frac{\|\{\boldsymbol{v} \in V | \boldsymbol{v} \in V'\}\|}{\|V\|}. \tag{9}$$

The higher the value of r, the better approximation. Note that we regard the convex hull obtained by Qhull as the exact convex hull, in this evaluation.

However, this metric alone would not be enough to assess the approximation quality. As an extreme example, if only one point of the approximate convex hull is wrong and that point is far away from the exact convex hull as shown in Fig. 9, r takes a large value but it cannot be said to be good approximation. Therefore, we also evaluated the value μ, which is the maximum value of the shortest distances among the vertices and boundaries between the exact convex hull and the approximate hull as shown in Fig. 10. In addition, to remove the effect of size of convex hulls, the value of μ is normalized to the diameter of the point set P:

$$\mu = \frac{\max\left(\max_{\boldsymbol{v} \in V} d\left(\boldsymbol{v}, \partial CH'(P)\right), \max_{\boldsymbol{v'} \in V'} d\left(\boldsymbol{v'}, \partial CH(P)\right)\right)}{\mathrm{diam}(P)} \tag{10}$$

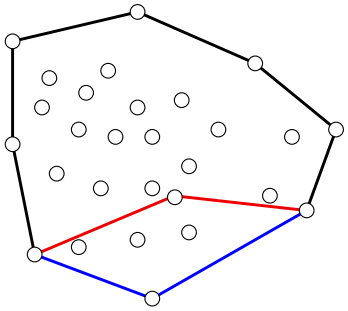


Fig. 9 Wrong detection example of approximate convex hull (red edges). The exact convex hull is with blue edges.

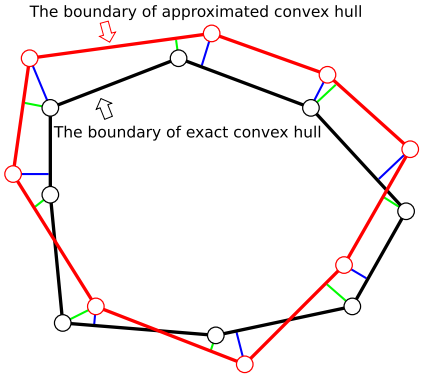


Fig. 10 Distance between boundaries and vertices.

Table 3 Resource usage and maximum operating frequency.

s	W_s	LUT	[%]	FF	[%]	BRAM	Fmax[MHz]
50	6	27364	5.09	38036	3.54	3.5	300.21
150	8	93572	17.41	116611	10.85	3.5	188.79
250	8	156096	29.04	198608	18.47	3.5	189.29
350	9	225054	41.86	283405	26.36	3.5	188.47

$$d\left(\boldsymbol{P}, \partial CH(P)\right) = \min_{0 \le k < h} d(\boldsymbol{P}, \overline{\boldsymbol{V_k}\boldsymbol{V_{k+1}}}) \tag{11}$$

$$d(\boldsymbol{P}, \overline{\boldsymbol{V_k}\boldsymbol{V_{k+1}}}) = \begin{cases} \|\boldsymbol{P} - \boldsymbol{V}_k\|, & (\alpha \le 0) \\ \|\boldsymbol{P} - \boldsymbol{V}_k - \alpha \boldsymbol{d}\|, & (0 < \alpha < l) \\ \|\boldsymbol{P} - \boldsymbol{V}_{k+1}\|, & (l \le \alpha) \end{cases} \tag{12}$$

where $\mathrm{diam}(P)$ is the diameter of the point set P, $\ell = ||\boldsymbol{V}_{k+1} - \boldsymbol{V}_k||$ is the length of the edge, $\boldsymbol{d} = (\boldsymbol{V}_{k+1} - \boldsymbol{V}_k)/\ell$ is the unit direction vector of the edge, and $\alpha = (\boldsymbol{P} - \boldsymbol{V}_k)^T \boldsymbol{d}$ is the signed distance between the projected point of $\boldsymbol{P}$ and $\boldsymbol{V}_k$. When $\mu \simeq 0$, the approximation can be considered to be good. In that case, the relative error η between the exact convex hull area S and the approximate convex hull area S':

$$\eta = \frac{|S - S'|}{S} \tag{13}$$

where

$$S = \frac{1}{2}\left|\sum_{k=0}^{h-1} \det\left(\boldsymbol{v_k} \;\; \boldsymbol{v_{k+1}}\right)\right| \tag{14}$$

$$S' = \frac{1}{2}\left|\sum_{k=0}^{h'-1} \det\left(\boldsymbol{v'_k} \;\; \boldsymbol{v'_{k+1}}\right)\right| \tag{15}$$

is also a useful evaluation metric.

4.1 Resource Usage and Execution Performance

The resource usage and the maximum operating frequency (Fmax) of this implementation are listed in Table 3. When $s = 50$, the resource usage was small and the maximum operating frequency was close to 300 MHz. When $s = 150$, 250, and 350, the maximum operating frequency was about 180 MHz, which was 100 MHz or more lower than that of $s = 50$. This is because the increase in the number of INNER_MINMAX circuits makes wiring congestion on the FPGA, resulting in an increase in wiring delays. Each time s was increased by 100, LUT utilization and FF utilization were increased by about 12 pt and 8 pt, respectively. Assuming a linear relationship between resource usage and s, the target FPGA (Virtex UltraScale xcvu095-ffva2104-2-e-es2) will be able to configure the system up to $s = 750$.

Since the size of each stack is $s+1$, the required memory capacity is $W_n \times (s+1)$ bits. Here, if $W_n = 25$ and $s = 350$, the required capacity becomes 8,775 bits, which corresponds to half of 0.5 BRAM (in this device, one BRAM can be utilized as two memory module each contains 18,000 bits). Therefore, as shown in Table 3, the total amount of BRAM remained at $0.5 \times 7 = 3.5$ regardless of s. This does not change even when s reaches 700, so BRAM usage does not limit the solvable problem size on this system.

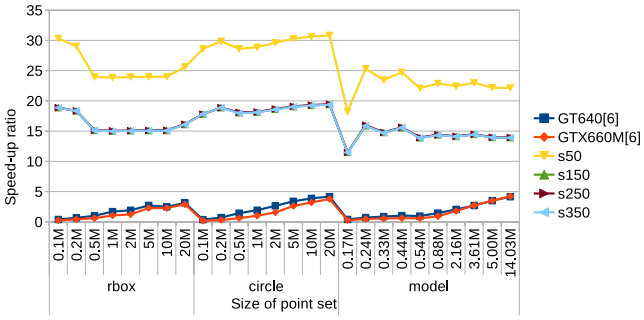


Fig. 11 Speedup ratios to Qhull execution.

Figure 11 shows the speedup ratios for CudaChain and our implementation to Qhull execution. Table 4 represents the execution times for implementations shown Fig. 11. When $s = 50$, the maximum operating frequency was high, and thus it was approximately 1.5 times faster than the designs with $s = 150$, 250, and 350. When $s = 150$, 250, and 350, there was almost no difference in terms of the maximum operating frequency and the execution time. This implementation is 18 to 30 times faster when $s = 50$, and 11 to 19 times faster when $s = 150$, 250, and 350 compared to Qhull. Compared to the implementation for $s = 50$, the performance was dropped when $s = 150$, 250 and 350. This is due to the FPGA implementation. An increase in s also increases the complexity of the FPGA circuit, resulting in the slower clock frequency as shown in Table 3. In addition, compared to CudaChain, it was more than 5 times faster when $s = 50$ and more than 3 times faster when $s = 150$, 250, and 350. Related work [8] achieved up to 14 times speedup to Qhull, and related work [10] achieved more than 10 times faster performance than Qhull. Since the point sets and Qhull versions used for evaluation differ from these GPU implementations in the literature, precise comparisons cannot be made, but at least we achieved higher performance than these GPU implementations.

4.2 Execution Efficiency

Table 5 shows power consumption, execution time, total energy, and energy efficiency for this method and GPU im-

Table 4 Execution times of Qhull, CudaChain and this method [ms].

			CudaChain [6]		This method			
Group	size	Qhull	GT640	GTX660M	s50	s150	s250	s350
rbox	0.1M	10.00	25.5	42.5	0.33	0.53	0.53	0.53
	0.2M	19.44	29.1	45.9	0.67	1.06	1.06	1.06
	0.5M	40.00	40.4	65.6	1.67	2.65	2.64	2.65
	1M	79.44	46.9	75.0	3.33	5.30	5.28	5.31
	2M	159.44	83.5	129.1	6.66	10.59	10.57	10.61
	5M	399.44	147.0	174.8	16.66	26.49	26.42	26.53
	10M	799.44	321.9	351.8	33.31	52.97	52.83	53.06
	20M	1705.00	544.4	587.4	66.62	105.94	105.66	106.12
circle	0.1M	9.44	25.9	54.0	0.33	0.53	0.53	0.53
	0.2M	20.00	28.3	65.5	0.67	1.06	1.06	1.06
	0.5M	47.78	33.3	78.1	1.67	2.65	2.64	2.66
	1M	96.11	50.0	95.0	3.33	5.30	5.29	5.31
	2M	197.22	74.6	126.8	6.66	10.60	10.57	10.62
	5M	504.44	148.6	193.0	16.66	26.49	26.42	26.53
	10M	1020.00	263.2	317.9	33.31	52.97	52.83	53.06
	20M	2053.89	492.8	543.6	66.62	105.94	105.66	106.12
model	0.17M	10.56	26.5	39.7	0.58	0.92	0.92	0.92
	0.24M	20.00	28.0	41.6	0.79	1.26	1.25	1.26
	0.33M	25.56	29.6	45.4	1.09	1.73	1.73	1.74
	0.44M	36.11	35.1	59.8	1.46	2.32	2.31	2.32
	0.54M	40.00	42.1	68.7	1.81	2.88	2.87	2.89
	0.88M	67.22	46.3	73.9	2.94	4.68	4.67	4.69
	2.16M	161.11	78.5	90.6	7.18	11.42	11.39	11.44
	3.61M	276.11	102.5	101.7	12.02	19.12	19.07	19.15
	5.00M	369.44	105.5	106.0	16.66	26.49	26.42	26.53
	14.03M	1033.89	248.8	245.2	46.73	74.31	74.11	74.43

Table 5 Performance and efficiency comparison.

			Power [W]	Exec. time [ms]		Total energy [J]		Efficiency [M points/J]	
	Platform			2M	10M	2M	10M	2M	10M
2009 [9]	GTX 280		236	† 30.0	†180.0	7.08	42.48	0.28	0.24
2011 [8]	GTX 280		236	† 30.0	†130.0	7.08	30.68	0.28	0.33
2012 [10]	GTX 260		182	† 95.0	-	17.29	-	0.12	-
2016 [6]	GTX 660M + i7-3610QM		95	129.1	351.8	12.26	33.42	0.16	0.30
	GT 640 + i5-3470		142	83.5	321.9	11.86	45.71	0.17	0.22
2020 [20]	GT 640 + i5-3470		142	44.4	163.4	6.30	23.20	0.32	0.43
	Quadro M5000 + Xeon E5-2650v3		255	33.4	93.6	8.52	23.87	0.23	0.42
	Quadro M5000 × 2 + Xeon E5-2650v3		405	44.6	83.3	18.06	33.74	0.11	0.30
2020 [21]	GTX 1650		75	-	210.2	-	15.76	-	0.63
This method	Virtex UltraScale xcvu095	s50	18	6.7	33.3	0.12	0.60	16.58	16.68
		s350	20	10.6	53.1	0.21	1.06	9.43	9.42

plementation in related work. Here, the energy efficiency was defined as the number of points processed per unit energy consumption. Our design was implemented on a Xilinx VCU108 evaluation board equipped with a Virtex UltraScale xcvu095 FPGA and the power consumption of the entire board was measured by a watt checker plugged in to the AC power line. On the other hand, the power consumption for the GPU implementation was estimated from the thermal design power (TDP) of the devices.

Although the performance has not been reported in the literature, the latest GPUs (such as RTX3080) might outperform the FPGA implementation. However, the latest GPU consumes as much as 320 W. On the other hand, the FPGA implementation on the VCU108 evaluation board was about 20 W. One of the advantages of the FPGA approach is a high degree of energy efficiency with a relatively slow clock frequency.

4.3 Accuracy Evaluation

Since the proposed approach is an approximate algorithm, accuracy of results is an important factor. When two adjacent scan vector S_0 and S_1 detect two convex hull vertices V_i and V_{i+1}, let C_i be a cross point of the two perpendicular line as shown in Fig. 12. If other convex hull vertices exist inside the triangle $V_iV_{i+1}C_i$, they will be overlooked since the triangle is a "shadow" of the projection. Qualitatively, an increase in the number of scan vectors s will increase the accuracy by reducing the angle between the two adjacent scan vectors and the shadow area. However, the probability of overlook largely depends on the shape of the convex hull and thus

†Estimated value from the graph.

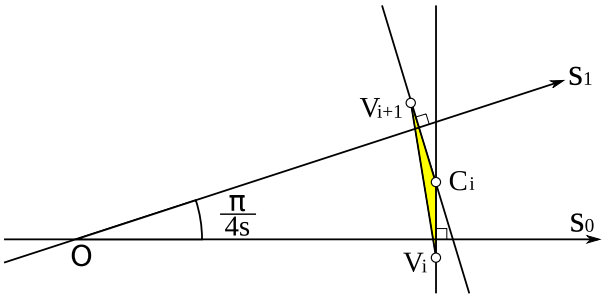


Fig. 12 Missing point area

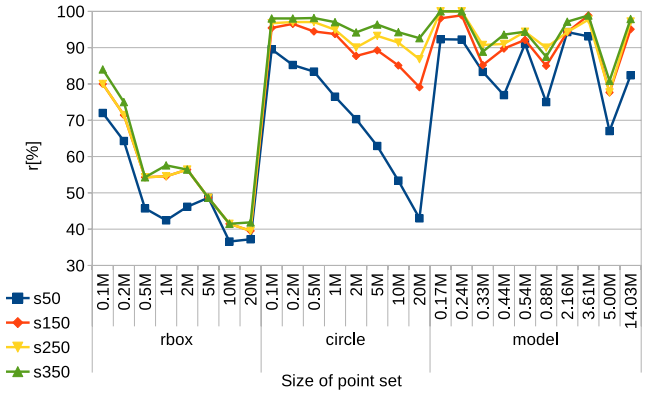


Fig. 13 Evaluation results for approximation metric: r.

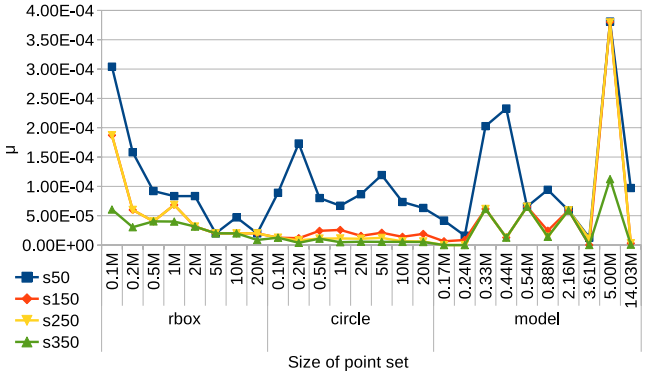


Fig. 14 Evaluation results for approximation metric: μ.

general formal modeling is not straightforward. Therefore, we experimentally performed the accuracy evaluation.

Figures 13, 14, and 15 show the evaluation results of approximation metrices r, μ, and η, respectively. Only four approximate convex hulls achieved r of 100%. Especially, r values for the rbox group were relatively low, ranging 36.5% to 84%. However, the maximum μ value was 3.809e-04. This means that the maximum difference between the polygons is 0.038% of the diameter, which is acceptable in most practical applications. Furthermore, the value of η, the maximum value of the relative error in the area, was only 3.770e-04. These results suggest that the proposed algorithm can be used to find appropriate convex polygons to roughly grab abstract object shapes in applications such as collision detection. The largest error was observed when $s = 50$ for

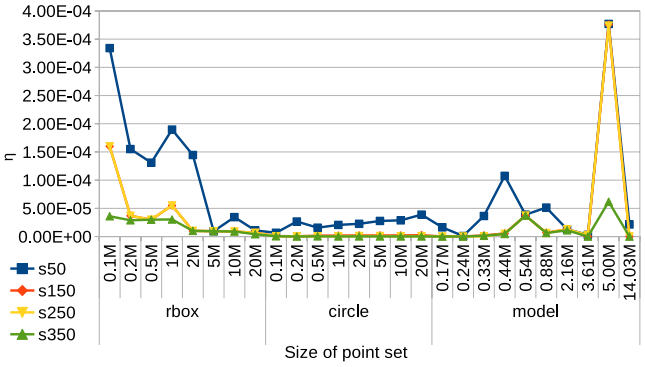


Fig. 15 Evaluation results for approximation metric: η.

all the point sets. The error decreases as s increases, and the smallest error was shown when $s = 350$ for most point sets.

In this paper, we discussed only for 2D cases. For three or larger dimensions, theoretically, the vertices of the convex hull can be computed in $O(n+s)$. However, as the dimension increases, the hardware size for inner product arithmetic and coordinate registers will increase almost linearly to the dimension size. As a result, the hardware amount will limit the degree of parallelism and performance. In addition, the resource limitation also severely affects the accuracy of results, since more scan vectors are required in larger dimension spaces.

5. Conclusion

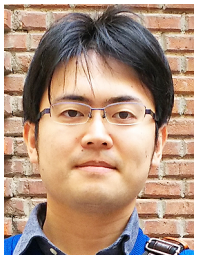
In this paper, we proposed an approximate convex hull algorithm, which is oriented for hardware implementation in a sense that input points can be processed in a pipelined manner without storing all the points in memory. Then, the proposed algorithm was implemented on FPGA without external memory and the performance and accuracy were evaluated. Unlike the related work such as [11], our approach does not need to sort the input point set in advance. In addition, by changing the design parameter s, which is a discrete resolution of a scan vector, different trade-off points among hardware resources, performance, and approximation quality can be selected depending on application requirements. In the case of $s = 50$, up to 30 times performance was achieved, compared to the software library Qhull. The maximum approximation error metrices μ and η were only 3.809e-04 and 3.770e-04, respectively. The evaluation results suggest our approach is effective in many practical real-time application domains. Expansion of the proposed method to 3D convex hull problems is one of our interesting future work.

References

[1] T. Mori, T. Manabe, and Y. Shibata, "Fast and memory efficient approximated convex hull computation with FPGA," Proc. International Symposium on Computing and Networking (CANDAR), 2020.
[2] S. Ramaswami, "Convex hulls: Complexity and applications (a survey)," Technical Report, University of Pennsylvania, 1993.
[3] "Qhull," http://qhull.org/, accessed March 1, 2021.
[4] W. Eddy, "A new convex hull algorithm for planar sets," ACM Trans.

Math. Softw., vol.3, no.4, pp.398–403, 1977.
[5] C.B. Barber, D.P. Dobkin, and H.T. Huhdanpaa, "The quickhull algorithm for convex hulls," ACM Trans. Math. Softw., vol.22, no.4, pp.469–483, 1996.
[6] G. Mei, "CudaChain: An alternative algorithm for finding 2D convex hulls on the GPU," SpringerPlus, vol.5, pp.1–26, 2016.
[7] A.A. Melkman, "On-line construction of the convex hull of a simple polyline," Inform. Process. Lett., vol.25, no.1, pp.11–12, 1987.
[8] S. Srungarapu, D.P. Reddy, K. Kothapalli, and P.J. Narayanan, "Fast two dimensional convex hull on the GPU," International Conference on Advanced Information Networking and Applications Workshop, pp.7–12, 2011.
[9] D. Srikanth, P. Reddy, K. Kothapalli, R. Govindarajulu, and P.J. Narayanan, "Parallelizing two dimensional convex hull on NVIDIA GPU and Cell BE," International Conference on High Performance Computing, pp.1–5, 2009.
[10] S. Tzeng and J.D. Owens, "Finding convex hulls using Quickhull on the GPU," arXiv preprint arXiv:1201.2936, 2012.
[11] K. Kanazawa, K. Kemmotsu, Y. Mori, N. Aibe, and M. Yasunaga, "High-speed calculation of convex hull in 2D images using FPGA," Proc. International Conference on Parallel Computing (ParCo), pp.532–542, 2015.
[12] A.M. Andrew, "Another efficient algorithm for convex hulls in two dimensions," Inform. Process. Lett., vol.9, no.5, pp.216–219, 1979.
[13] R.L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," Inform. Process. Lett., vol.1, no.4, pp.132–133, 1972.
[14] L. Kavan, I. Kolingerova, and J. Zara, "FAST approximation of convex hull," The 2nd International Association of Science and Technology for Development International Conference on Advances in Computer Science and Technology, pp.101–104, 2006.
[15] J.L. Bentley, G.M. Faust, and F.P. Preparata, "Approximation algorithms for convex hulls," Commun. ACM, vol.25, no.1, pp.64–68, 1982.
[16] Z. Hussain, "A fast approximation to a convex hull," Pattern Recogn. Lett., vol.8, no.5, pp.289–294, 1988.
[17] M.Z. Hossain and M.A. Amin, "On constructing approximate convex hull," American Journal of Computational Mathematics, vol.3, no.1A, pp.11–17, 2013.
[18] "The Standford 3D Scanning Repository," http://www.graphics.stanford.edu/data/3Dscanrep/, accessed July 2, 2021.
[19] "GIT Large Geometry Models Archive," https://www.cc.gatech.edu/projects/large_models/, accessed July 2, 2021.
[20] J. Qin, G. Mei, S. Cuomo, S. Guo, and Y. Li, "CudaCHPre2D: A straightforward preprocessing approach for accelerating 2D convex hull computations on the GPU," Concurrency and Computation Practice and Experience, vol.32, no.4, pp.1–12, 2019.
[21] "Computing the convex hull on GPU," https://timiskhakov.github.io/posts/computing-the-convex-hull-on-gpu, accessed July 2, 2021.

Tatsuma Mori graduated from Nagasaki University in 2020. Now he is a master student at the Graduate School of Engineering, Nagasaki University. His research interests include the real-time processing with an FPGA.



Taito Manabe received the B.E. and M.E. degrees from Nagasaki University, Japan, in 2016 and 2018, respectively. Now he is a doctoral student at the Graduate School of Engineering, Nagasaki University, and is pursuing Ph.D. degree. His research interests include real-time processing with an FPGA.

Yuichiro Shibata received the B.E. degree in electrical engineering, the M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is a professor at School of Information and Data Sciences, Nagasaki University. He was a Visiting Scholar at University of South Carolina in 2006. His research interests include reconfigurable systems and parallel processing. He received the Best Paper Award of IEICE in 2004.