Entre Ciencia e Ingeniería, vol. 16, no. 32, julio-diciembre de 2022, páginas 27-34. DOI: https://doi.org/ 10.31908/19098367.2668
ISSN 1909-8367 (Impreso), ISSN 2539-4169 (En línea)

# Implementación de algoritmos para calcular el Convex Hull [1]

# Implementation of algorithms to compute the Convex Hull

C. A. Candela, L. E. Sepúlveda, J. C. Chavarro, C. A. Meneses, J. A. Sanabria y O. Arcila

Recibido: julio 15 de 2021 – Aceptado: diciembre 30 de 2022

[1]***Resumen*— La geometría computacional es una disciplina enfocada en la resolución de problemas en el ámbito geométrico. En este contexto, el algoritmo para calcular el polígono convexo llamado *Convex Hull* (*CH*) es importante, debido a que es la base de muchos otros algoritmos. El objetivo de la investigación fue implementar algoritmos que calculan el *CH* incorporando modificaciones para reducir el tiempo de ejecución. El trabajo inició con la revisión bibliográfica acerca de geometría computacional y los algoritmos destacados en el cálculo del *CH*. Posteriormente, se realizó la implementación en *JAVA* de los algoritmos *QuickHull, Gift Wrapping* y *Graham Scan* en sus versiones originales; también se implementaron algunas versiones con modificaciones. Al finalizar la implementación, se ejecutaron pruebas para verificar los tiempos de ejecución. Finalmente, se comprobó que el algoritmo *QuickHull* es el más rápido entre las implementaciones realizadas en esta investigación. También se nota reducción en los tiempos de ejecución en las implementaciones modificadas con relación a las originales de los algoritmos *Gift Wrapping* y *Graham Scan*.**

***Palabras clave*— Geometría computacional, Covex Hull, Gift Wrapping, Graham Scan, QuickHull.**

***Abstract*— Computational geometry is a discipline focused on solving problems in the geometric domain. In this context, the algorithm for computing the convex polygon called *Convex Hull (CH)* is important, because it is the basis for many other algorithms. The objective of the research was to implement algorithms that compute the *CH* incorporating modifications to reduce the execution time. The research started with a bibliographic review of computational geometry and the algorithms highlighted in the calculation of *CH*. Subsequently, the *QuickHull, Gift Wrapping,* and *Graham Scan* algorithms were implemented in *JAVA* in their original versions; some versions with modifications were also implemented. Upon completion of implementation, tests were run to verify the execution times. Finally, the *QuickHull* algorithm was found to be the fastest among the implementations performed in this research. It is also noted a reduction in execution times in the modified implementations in relation to the original ones of the *Gift Wrapping* and *Graham Scan* algorithms.**

***Keywords*— Computational Geometry, Convex Hull, Gift Wrapping, Graham Scan, QuickHull.**

## I. Introducción

La *CG (Computacional Geometry)* puede entenderse como una disciplina que se ocupa del análisis y diseño de algoritmos computacionales para resolver problemas en el ámbito geométrico [1], [2]. Muchos de estos algoritmos utilizan como base una cadena poligonal convexa comúnmente denominada *CH* (*Convex Hull)*. Típicamente, el *CH* es considerado como un importante problema computacional y se refiere a la construcción de polígono convexo que constituye la envoltura más externa de un conjunto de puntos dado [2].

Realizar la implementación de algoritmos para obtener una cadena poligonal correspondiente al *CH*, es un cálculo computacional con diversas aplicaciones como son la planificación de rutas evitando colisiones (por ejemplo en robots o vehículos autónomos [3], [4]) y el reconocimiento de patrones en gráficos (por ejemplo en la visión por computador [5]). En este sentido, el presente trabajo implementó los algoritmos *QuickHull*, *Gift Wrapping* y *Graham Scan* que calculan el *CH* incorporando modificaciones para reducir el tiempo de ejecución, lo anterior, enmarcados en la investigación base que contempla entre otros aspectos reducir la latencia en el intercambio de mensajes entre microservicios

[1]Producto derivado del proyecto de investigación "Adaptación de un modelo transaccional extendido para la implementación de transacciones de negocio en microservicios", apoyado por la Universidad Tecnológica de Pereira a través del Doctorado en Ingeniería.

C. A. Candela, Universidad Tecnológica de Pereira, Universidad del Quindío, Colombia, email: christiancandela@uniquindio.edu.co.

L. E. Sepúlveda, Universidad Tecnológica de Pereira, Universidad del Quindío, Colombia, email: lesepulveda@uniquindio.edu.co.

J. C. Chavarro, Universidad Tecnológica de Pereira, Colombia, email: jchavar@utp.edu.co.

C. A. Meneses, Universidad Tecnológica de Pereira, Colombia, email: cmeneses@utp.edu.co.

J. A. Sanabria, Universidad del Valle, Colombia, email: john.sanabria@correounivalle.edu.co.

O. Arcila, email: olarguz@gmail.com.

**Como citar este artículo:** C.A. Candela, L. E. Sepúlveda, J. C. Chavarro, C. A. Meneses, J. A. Sanabria, O. Arcila. Implementación y pruebas de algoritmos para calcular el Convex Hull, Entre Ciencia e Ingeniería, vol. 16, no. 32, pp.27-34 julio-diciembre 2022. DOI: https://doi.org/10.31908/19098367.2668.

[6].

El contenido adicional de este trabajo consta de las siguientes cinco secciones: *II. Trabajo relacionado, III. Marco teórico, IV. Metodología, V. Resultados y Análisis* y *VI. Conclusiones*.

## II. Trabajo relacionado

Esta sección presenta cronológicamente algunos estudios relacionados con la presente investigación.

*Sunday* en su trabajo del 2006 denominado "*The Convex Hull of a 2D Point Set or Polygon*" [7], presenta una revisión general de diferentes algoritmos para el cálculo del *CH*.

Para 2009, *Qiha* en la publicación denominada "*New serial and parallel algorithms for finding convex hull based on clusters, domains and directions from single to multitude*" [8], presenta un teorema fundamental isomórfico a partir del cual propone dos algoritmos (uno serial y otro paralelo) para encontrar el *CH*.

*Chiang* en su trabajo del 2010 titulado "*Solving 2D Convex Hull With Parallel Algorithms*" [9], presenta una implementación de un algoritmo divide y vencerás que calcula el *CH* usando paralelismo mediante el aprovechamiento de CPU y GPU.

En 2015, *Hoang* y *Linh* en su trabajo "*Quicker than Quickhull*" [10], proponen una variación del algoritmo *QuickHull* denominado *QuickerHull*; el cual incluye mejoras para reducir el tiempo de ejecución del algoritmo.

*Skala et al.* en su trabajo de 2016 llamado "*Reducing the number of points on the convex hull calculation using the polar space subdivision in E2*" [11], presentan un método basado en una subdivisión del espacio polar para reducir el conjunto de puntos de un polígono dado, con el fin de acelerar los algoritmos para el cálculo del *CH*.

En 2018, *Gamby* y *Katajainen* realizaron implementaciones de los algoritmos *Plane-Sweep*, *Torch*, *QuickHull* y *Throw-Away* para evaluar su eficiencia, los resultados fueron publicados en "*Convex-Hull Algorithms: Implementation, Testing, and Experimentation*" [12]. En 2019, estos mismos autores, en su trabajo "*A Faster Convex-Hull Algorithm via Bucketing*" [13], propusieron una variante del algoritmo *Via Bucketing*, logrando una mejora en su tiempo de ejecución.

*Borna* en su publicación de 2019 titulada "*Sweep Line Algorithm for Convex Hull Revisited*" [14], presenta un nuevo algoritmo para el cálculo del *CH* denominado *Sweep Line*, logrando un tiempo de ejecución en el orden de *O(nlogn)*.

Finalmente, *Lesjak* en 2021 con su trabajo "*Pregled in primerjava algoritmov za izračun konveksne ovojnice*" [15] presenta una revisión de algoritmos clásicos y modernos para el cálculo del *CH*. En el trabajo se destaca el algoritmo *Ordered Hull* siendo más rápido que el *Quickhull*.

## III. Marco teórico

Esta sección contiene aspectos teóricos relevantes para la comprensión del tema tratado en este trabajo.

### A. Geometría computacional

La *CG* es un campo de la informática que comenzó en los años 70s y según *Forrest*, citado por [16] está relacionada con la representación computarizada, análisis, síntesis (diseño) y manufactura controlada por computadora de formas bidimensionales y tridimensionales. En otras palabras, la *CG* puede entenderse como una disciplina que se ocupa del análisis y diseño de algoritmos computacionales para resolver problemas en el ámbito geométrico. La *CG* ha tenido una fuerte interacción con diversos campos de ciencia e ingeniería como *Algoritmos*, *Estructuras de Datos*, *Matemáticas*, *Geometría Euclidiana*, *Optimización*, entre otros [17]. Desde la década de los 90s existe una demanda de la *CG* en una variedad de áreas de las ciencias aplicadas tales como *Sistemas de Información Geográfica*, *Visualización*, *Robótica* y *Gráficos por Computadora*; estas áreas han propiciado aún más la investigación en *CG* [17].

*1) Términos básicos*

*a) Punto: Es considerado el elemento geométrico más sencillo. Definido como un conjunto de números reales ya sean en dos [x,y] o tres [x,y,z] dimensiones. En la Fig. 1, un punto es cualquier de los elementos del conjunto Z.*

*b) Segmento: Se define a partir de dos puntos [ punto$_1$, punto$_2$]; si se llega a considerar la orientación del segmento se le considera un vector. En la Fig. 1 se muestra un segmento conformado por los puntos pq.*

*c) Polígono: Se denomina polígono a una superficie plana cuyo borde está formado por segmentos rectos. En la Fig. 1 se muestra el polígono CH.*

*d) Cadena poligonal: Un conjunto ordenado de puntos distintos en el plano determina una cadena poligonal, que es la unión de los segmentos. CH de en la Fig. 1, también representa una cadena poligonal.*

*e) Convexidad: El concepto de convexidad puede definirse así: un conjunto S del plano se considera convexo si j Î S y k Î S, lo que implica que el segmento jk Í S [18], [19], ver Fig. 2.*

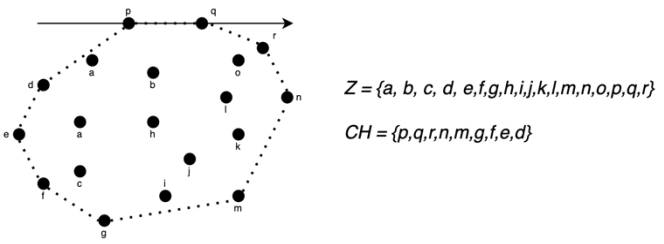


Fig. 1. Representación del Convex Hull [18].

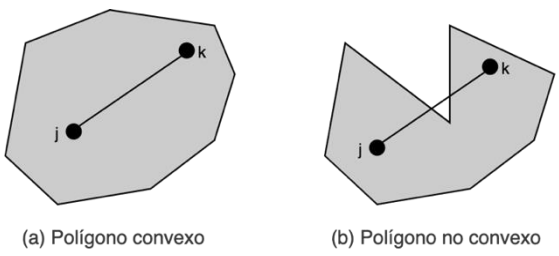


Fig. 2. Representación de un polígono convexo y no convexo [18],[19].

*2) Cálculo del Convex Hull*

Es importante reconocer que el *CH* de un conjunto *Z* (conjunto de puntos en el plano), es un polígono convexo que contiene a *Z* [18]. Yendo más allá, y basados en la Fig. 1, al identificar un borde del *CH* del polígono *Z* conformado por los puntos *pq* notamos que ambos son puntos de *Z* y que al trazar una línea a través de estos puntos (teniendo en cuenta para la orientación el sentido de las manecillas del reloj) obtendremos que los demás puntos de *Z* se encuentran a la derecha. Esta es la idea que abarca el aspecto geométrico del problema relacionado con el cálculo del *CH*.

Existen diversos algoritmos para calcular el *CH* siendo algunos de los más destacados los siguientes: *QuickHull*, *Gift Wrapping* y *Graham Scan*. A continuación, se presenta el pseudocódigo de cada uno de ellos.

*a) Algoritmo QuickHull*

Este algoritmo fue sugerido por diversos investigadores a finales de los 70s [20], [21], y fue llamado *QuickHull* por *Preparata* y *Shamos* en 1985 [16], citado por [19], [22] y [23]. La complejidad del *QuickHull* según [20] es del orden *O(nh)*, siendo *n* el número de puntos y *h* el número de vértices del *CH*. Ver Fig. 3.

*b) Algoritmo Gift Wrapping*

Este algoritmo fu propuesto por *Chand* y *Kapur* en 1970 [24] y tiene una complejidad del orden *O(nh)*, siendo *n* el número de puntos y *h* el número de vértices del *CH*. Ver Fig. 4.

*c) Algoritmo Graham Scan*

Este algoritmo hereda el nombre del apellido de su autor *Ronald Graham*, quien lo propuso en 1972 [25]. Este algoritmo tiene una complejidad del orden *O(nlogn)*, siendo *n* el número de puntos. Ver Fig. 5.

**Datos:** Nube de puntos
**Resultado:** Cadena poligonal correspondiente al *CH*
**Función** $QuickHull(a, b, S)$
  **si** $S = 0$ **entonces**
    $retornar()$
  **en otro caso**
    $c =$ punto mas distantes desde $\overline{ab}$
    $A =$ puntos estrictamente a la derecha de $(a, c)$
    $B =$ puntos estrictamente a la derecha de $(c, b)$
    $retornar(QuickHull(a, c, A) + (c) +$
     $QuickHull(c, b, B))$
  **fin**
**fin**

Fig. 3. Algoritmo *QuickHull*.

**Datos:** Nube de puntos
**Resultado:** Cadena poligonal correspondiente al *CH*
**repita**
  **para cada** $j \neq i$ **hacer**
    Calcule ángulo antihorario $\Theta$ desde borde anterior del casco;
    Sea $k$ el índice del punto con el menor $\Theta$;
    Salida $(Pi, Pk)$ como borde del *hull*;
    $i = k$;
  **fin**
**mientras** $(i = i_0)$;

Fig. 4. Algoritmo *Gift Wrapping*.

**Datos:** Nube de puntos
**Resultado:** Cadena poligonal correspondiente al *CH*
$i = 2$;
$PilaS = (p_1, p_0) = (p_t, p_{t-1})$ ; $t$ es índice tope de pila;
**mientras** $i < n$ **hacer**
  **si** $p_i$ *está estrictamente a la izquierda de* $p_{t-1}p_t$ **entonces**
    $Push(p_i, S)$;
    $i = i + 1$;
  **en otro caso**
    $Pop(S)$ ;
  **fin**
**fin**

Fig. 5. Algoritmo *Graham Scan*.

## IV. Metodología

El proceso metodológico seguido en este trabajo consistió en la implementación de los siguientes pasos:

### *A. Paso No 1 - Implementación de los algoritmos*

Se implementó en lenguaje de programación *JAVA* algoritmos para el cálculo del *CH*. Además, se construyeron versiones alternativas a los algoritmos buscando mejorar los tiempos de ejecución.

### *B. Paso No 2 - Generación de los datos de prueba*

En este paso se generaron veinte archivos, cada uno con un conjunto de puntos aleatorios, cuyos valores coordenadas (x,y) están entre -10.000 y 10.000. En la generación de los puntos se usaron algoritmos con una distribución uniforme, lo anterior para que todos los números entre -10000 y 10000 tuvieran la misma probabilidad de ocurrencia. Por ejemplo, si se generan 20.000 valores aleatorios, debería esperarse que cada valor se genere aproximadamente 1 vez en promedio. Sin embargo, debido a la naturaleza aleatoria de la función empleada, el número real de veces que se genera cada valor puede variar ligeramente. En este caso, la probabilidad de que se genere un número determinado es de 1/20001, porque hay 20001 valores posibles en el intervalo (de -10000 a 10000, ambos inclusive). El número de puntos de cada uno de los archivos varia así: 5.000, 10.000, 15.000, 20.000, 25.000,

30.000, 35.000, 40.000, 45.000, 50.000, 55.000, 60.000, 65.000, 70.000, 75.000, 80.000, 85.000, 90.000, 95.000 y 10.000.

### C. Paso No 3 - Ejecución de los algoritmos

Se ejecutó los algoritmos en un computador portátil con procesador Core i7 de 4 núcleos a 2.3 GHz, 16 Gigas de memoria RAM DDR3 a 1600 MHz, unidad de estado sólido WDC WDS100T2B0A y tarjetas de video Intel HD Graphics 40000 y NVIDIA GeForce GT 650M. Para la ejecución de cada uno de los algoritmos se usó como parámetro de entrada cada uno de los veinte archivos generados. Cómo estrategia para disminuir el ruido generado por otros procesos ejecutados en el computador, cada uno de los archivos se procesa con cada uno de los algoritmos 100 veces; esto permitió tomar los tiempos para cada algoritmo al calcular el *CH* para los puntos dados.

### D. Paso No 4 - Consolidación de resultados

Se calculó el tiempo promedio de las 100 ejecuciones para cada uno de los archivos de puntos en cada algoritmo.

### E. Paso No 5 - Análisis de los resultados

Se tabularon los tiempos promedio calculados con la cantidad de puntos correspondiente a cada ejecución para cada uno de los algoritmos, se construyeron las gráficas y se compararon los resultados obtenidos.

## V. RESULTADOS Y ANÁLISIS

A continuación, se presentan los resultados de la implementación y ejecución de los algoritmos *QuickHull*, *Gift Wrapping* y *Graham Scan*.

### A. Implementación del QuickHull

La Tabla I muestra los resultados obtenidos al ejecutar este algoritmo con diferentes conjuntos de puntos. Además, en la imagen Fig. 6 se puede observar un comportamiento coherente de los tiempos de ejecución con relación a la complejidad del algoritmo. Sin embargo, al inicio y fin del gráfico, se observan datos atípicos que pueden deberse a la falta de asepsia computacional.

TABLA I.
TIEMPOS DE EJECUCIÓN OBTENIDOS AL EJECUTAR EL ALGORITMO QUICKHULL.

| Cantidad de puntos | Milisegundos | Cantidad de puntos | Milisegundos |
|---|---|---|---|
| 5.000 | 2 | 55.000 | 28 |
| 10.000 | 34 | 60.000 | 35 |
| 15.000 | 8 | 65.000 | 39 |
| 20.000 | 10 | 70.000 | 36 |
| 25.000 | 12 | 75.000 | 41 |
| 30.000 | 15 | 80.000 | 51 |
| 35.000 | 18 | 85.000 | 47 |
| 40.000 | 21 | 90.000 | 52 |
| 45.000 | 23 | 95.000 | 60 |
| 50.000 | 26 | 100.000 | 134 |

### B. Implementación del Gift Wrapping

Durante la implementación del algoritmo *Gift Wrapping*, se construyeron dos versiones, una original y otra modificada.

#### 1) Gift Wrapping - Versión Original

Implementación original del algoritmo *Gift Wrapping* descrito en el trabajo [24] (ver Fig. 4). La tabla II muestra los tiempos obtenidos al ejecutar el algoritmo. Además, en la Fig. 7 se muestra el comportamiento del algoritmo, que es consistente a su complejidad.

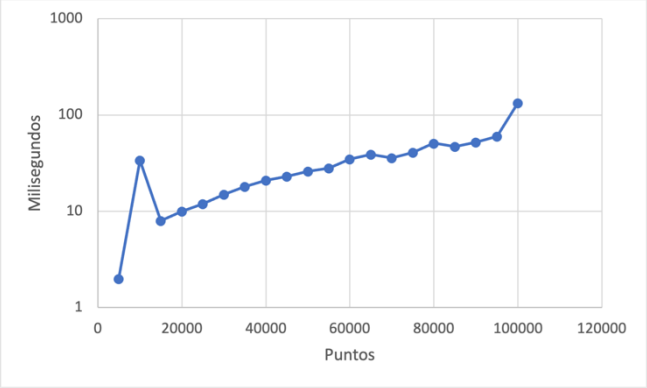


Fig. 6. Comportamiento del algoritmo *QuickHull* según la Tabla I.

TABLA II.
TIEMPOS DE EJECUCIÓN OBTENIDOS AL EJECUTAR EL ALGORITMO GIFT WRAPPING – VERSIÓN ORIGINAL.

| Cantidad de puntos | Milisegundos | Cantidad de puntos | Milisegundos |
|---|---|---|---|
| 5.000 | 38 | 55.000 | 455 |
| 10.000 | 144 | 60.000 | 594 |
| 15.000 | 158 | 65.000 | 804 |
| 20.000 | 218 | 70.000 | 914 |
| 25.000 | 257 | 75.000 | 795 |
| 30.000 | 217 | 80.000 | 991 |
| 35.000 | 485 | 85.000 | 1011 |
| 40.000 | 423 | 90.000 | 965 |
| 45.000 | 481 | 95.000 | 946 |
| 50.000 | 546 | 100.000 | 1418 |

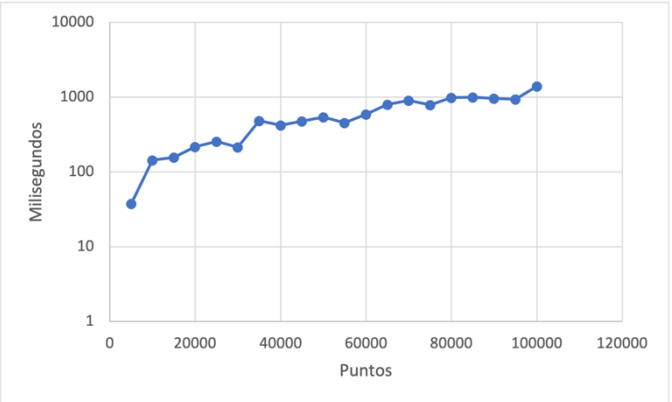


Fig. 7. Comportamiento del algoritmo Gift Wrapping - Versión Original según la Tabla II.

### C. Gift Wrapping - Versión Modificada

Esta implementación corresponde a una versión modificada del algoritmo *Gift Wrapping* original y continúa teniendo la misma complejidad *O(nh)*.

La tabla III muestra una reducción significativa en los

tiempos de ejecución en comparación con los datos del algoritmo original (ver tabla II). Esta reducción en los tiempos de ejecución está relacionada con la estrategia usada para comparar los ángulos formados por los vértices. Además, en la Fig. 8 se muestra el comportamiento del algoritmo *Gift Wrapping - Versión modificada*. Sin embargo, al inicio del gráfico, se observa un valor atípico que pueden deberse a la falta de asepsia computacional.

### *D. Comparación entre implementaciones del algoritmo Gif Wrapping*

En la Fig. 9 se muestra en una comparación entre las implementaciones del algoritmo *Gift Wrapping*. Como se puede observar, la versión modificada es más rápida que la versión original. La diferencia entre estas versiones radica en la estrategia usada en la comparación de los ángulos entre vértices, se puede inferir que la estrategia usada en el algoritmo modificado es efectiva, toda vez que evita realizar cálculos en la comparación de ángulos y en consecuencia se presenta una disminución en los tiempos de ejecución. La estrategia usada para comparar los ángulos en la versión original emplea la función del API de *JAVA* para calcular ángulos y así determinar cuál es el menor. Por otra parte, la estrategia usada por la versión modificada emplea la orientación de puntos para determinar cuál es el ángulo menor.

TABLA III.
TIEMPOS DE EJECUCIÓN OBTENIDOS AL EJECUTAR EL ALGORITMO GIFT WRAPPING – VERSIÓN MODIFICADA.

| Cantidad de puntos | Milisegundos | Cantidad de puntos | Milisegundos |
|---|---|---|---|
| 5.000 | 4 | 55.000 | 58 |
| 10.000 | 61 | 60.000 | 64 |
| 15.000 | 13 | 65.000 | 81 |
| 20.000 | 18 | 70.000 | 88 |
| 25.000 | 24 | 75.000 | 90 |
| 30.000 | 26 | 80.000 | 116 |
| 35.000 | 42 | 85.000 | 121 |
| 40.000 | 39 | 90.000 | 133 |
| 45.000 | 46 | 95.000 | 134 |
| 50.000 | 53 | 100.000 | 205 |

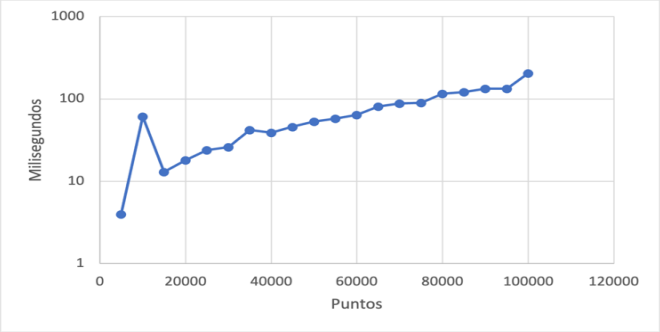


Fig. 8. Comportamiento del algoritmo *Gift Wrapping* - Versión Modificada según la tabla III.

### *E. Implementación del Graham Scan*

Durante la implementación del algoritmo *Graham Scan*, se construyeron tres versiones, que incluyen la versión original (ver Fig. 5) y dos versiones modificadas.

#### 1) Graham Scan - Versión Original

Implementación original del algoritmo Graham Scan descrito en el trabajo [25] (ver Fig. 5). La tabla IV muestra los tiempos obtenidos al ejecutar el algoritmo. Además, en la Fig. 10 se muestra el comportamiento del algoritmo, que es consistente a su complejidad.

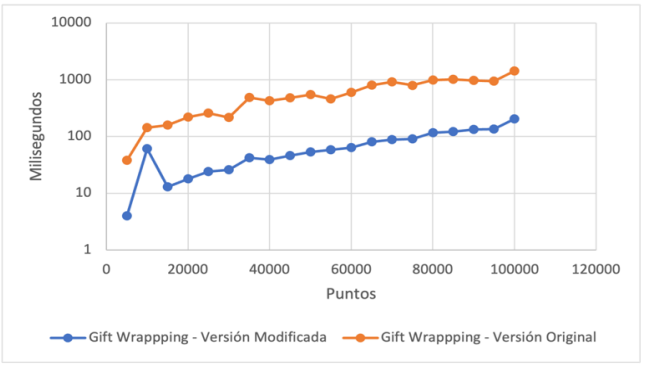


Fig. 9. Comparación del comportamiento entre la implementación de la versión original y la versión modificada del algoritmo *Gift Wrapping*.

TABLA IV.
TIEMPOS DE EJECUCIÓN EN OBTENIDOS AL EJECUTAR EL ALGORITMO GRAHAM SCAN – VERSIÓN ORIGINAL.

| Cantidad de puntos | Milisegundos | Cantidad de puntos | Milisegundos |
|---|---|---|---|
| 5.000 | 33 | 55.000 | 4171 |
| 10.000 | 148 | 60.000 | 4544 |
| 15.000 | 294 | 65.000 | 5344 |
| 20.000 | 513 | 70.000 | 6198 |
| 25.000 | 799 | 75.000 | 7130 |
| 30.000 | 1142 | 80.000 | 8053 |
| 35.000 | 1155 | 85.000 | 9236 |
| 40.000 | 2028 | 90.000 | 10263 |
| 45.000 | 2562 | 95.000 | 11351 |
| 50.000 | 3161 | 100.000 | 12650 |

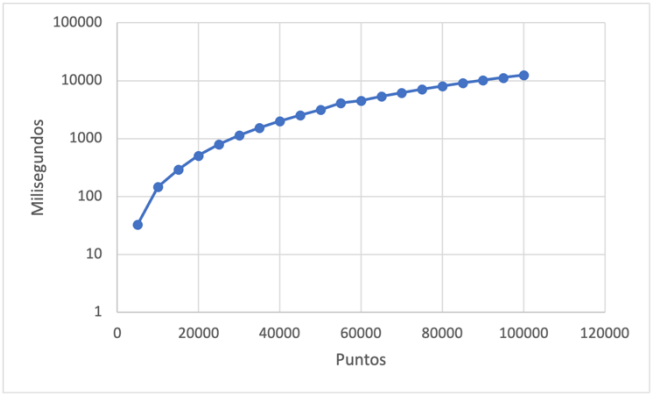


Fig. 10. Comportamiento del algoritmo *Graham Scan - Versión Original* y basado en los datos de la tabla IV.

#### *2) Graham Scan - Versión Modificada Nº 1*

Esta versión incorpora un proceso previo de eliminación de vértices, el cual consiste en encontrar los puntos extremos de la nube de puntos, para formar con ellos un rombo y remover de la nube de puntos todos aquellos que se encuentran al interior del rombo. Esta versión mantiene la complejidad original del algoritmo. Sin embargo, puede reducir drásticamente los tiempos de ejecución dependiendo de la cantidad de puntos que se remuevan al inicio.

La tabla V muestra una reducción significativa en los tiempos de ejecución en comparación con los datos del algoritmo original (ver tabla IV). Además, en la Fig. 11 se muestra el comportamiento del algoritmo *Graham Scan – Versión Modificada Nº 1*, que describe un comportamiento irregular. Esta situación está relacionada con la cantidad de puntos que logran eliminarse a través de la estrategia de reducción aplicada al inicio del algoritmo y la cual varía según la dispersión existente en la nube de puntos.

*3) Graham Scan - Versión Modificada Nº 2*

Esta versión incorpora un proceso previo de eliminación de vértices, el cual consiste en encontrar los puntos extremos de la nube de puntos, para formar con ellos un octágono y remover de la nube de puntos todos aquellos que se encuentran al interior. Esta versión mantiene la complejidad original del algoritmo. Sin embargo, puede reducir drásticamente los tiempos de ejecución dependiendo de la cantidad de puntos que se remuevan al inicio.

La tabla VI muestra una reducción significativa en los tiempos de ejecución en comparación con los datos del algoritmo original (ver tabla IV). Además, en la Fig. 12 se muestra el comportamiento del algoritmo *Graham Scan* en la versión modificada Nº 2 donde se evidencia que los tiempos de ejecución empleados indican una evidente optimización para el conjunto de puntos considerados en la investigación.

TABLA V.
TIEMPOS OBTENIDOS AL EJECUTAR EL ALGORITMO GRAHAM SCAN – VERSIÓN MODIFICADA Nº 1.

| Cantidad de puntos | Milisegundos | Cantidad de puntos | Milisegundos |
|---|---|---|---|
| 5.000 | 4 | 55.000 | 3408 |
| 10.000 | 20 | 60.000 | 148 |
| 15.000 | 83 | 65.000 | 894 |
| 20.000 | 137 | 70.000 | 1980 |
| 25.000 | 165 | 75.000 | 1961 |
| 30.000 | 598 | 80.000 | 1290 |
| 35.000 | 350 | 85.000 | 84 |
| 40.000 | 311 | 90.000 | 1368 |
| 45.000 | 667 | 95.000 | 544 |
| 50.000 | 834 | 100.000 | 3145 |

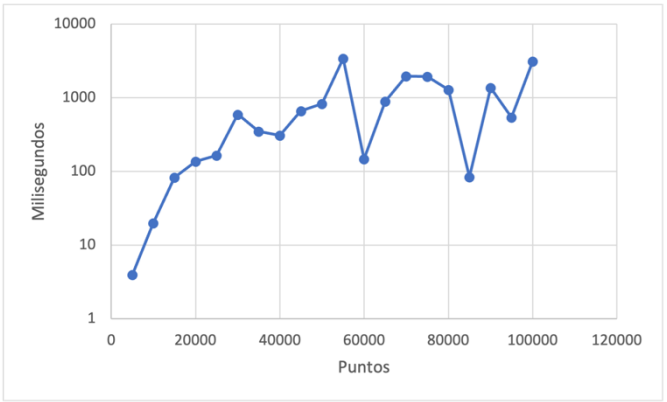


Fig. 11. Comportamiento del algoritmo *Graham Scan* - Versión Modificada Nº 1, según la tabla V.

*4)Comparación entre implementaciones del algoritmo Graham Scan*

En la Fig. 13 se muestra una comparación entre la implementación original del algoritmo *Graham Scan* y dos versiones modificadas. Se puede notar que las dos versiones modificadas presentan menor tiempo de ejecución que la versión original, siendo la versión modificada Nº 2 la más rápida. Aunque las versiones modificadas usan en general estrategias de reducción de puntos, la implementación da versión modificada Nº 2 demuestra ser más eficientes al lograr la eliminación de más puntos.

*5)Comparación de algoritmos QuickHull, Gift Wrapping y Graham Scan*

A continuación, se presentan comparaciones entre los algoritmos *QuickHull*, *Gift Wrapping* y *Graham Scan* en sus versiones originales (Ver Fig. 13). También se muestra el resultado de la comparación entre la implementación del algoritmo *QuickHull* y las versiones más rápidas de las implementaciones de los algoritmos *Gift Wrapping* y *Graham Scan* (ver Fig. 14).

TABLA VI.
. TIEMPOS OBTENIDOS AL EJECUTAR EL ALGORITMO GRAHAM SCAN – VERSIÓN MODIFICADA Nº 2.

| Cantidad de puntos | Milisegundos | Cantidad de puntos | Milisegundos |
|---|---|---|---|
| 5.000 | 3 | 55.000 | 51 |
| 10.000 | 11 | 60.000 | 52 |
| 15.000 | 12 | 65.000 | 58 |
| 20.000 | 16 | 70.000 | 65 |
| 25.000 | 21 | 75.000 | 73 |
| 30.000 | 25 | 80.000 | 77 |
| 35.000 | 28 | 85.000 | 80 |
| 40.000 | 33 | 90.000 | 94 |
| 45.000 | 40 | 95.000 | 96 |
| 50.000 | 43 | 100.000 | 105 |

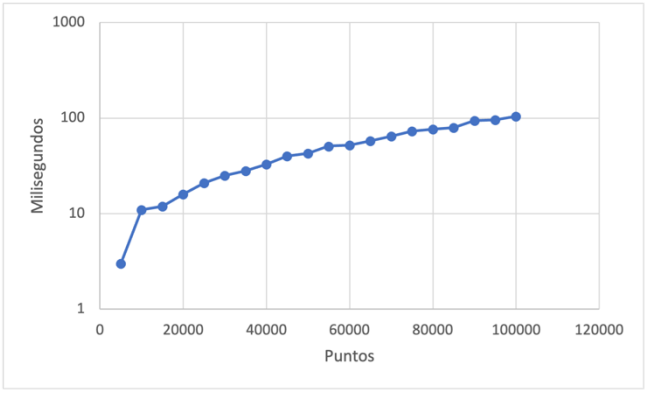


Fig. 12. Comportamiento del algoritmo *Graham Scan* - Versión Modificada Nº 2, según la tabla VI.

En la Fig. 13 se puede evidenciar que el *QuickHull* es el más rápido, marcando una notable diferencia con respecto a los tiempos de ejecución de los demás algoritmos.

Por otro lado, la Fig. 14 evidencia que el *QuickHull* sigue siendo el algoritmo más rápido, sin embargo, también se nota que la diferencia respecto a los tiempos de ejecución de los demás algoritmos se ha reducido significativamente. Esta reducción es debido a la inclusión de las modificaciones en los algoritmos *Gif Wrapping* y *Graham Scan*.

Considerando los tiempos observados para la ejecución de los algoritmos *Gift Wrapping* y *QuickHull* durante el

procesamiento del archivo con los 100 mil puntos. Esta situación puede haber sido influenciada por una distribución también atípica en los puntos, donde los algoritmos *QuickHull* y la versión modificada del *Gift Wrapping* fueron ligeramente más lento que la versión modificada del *Graham Scan.*

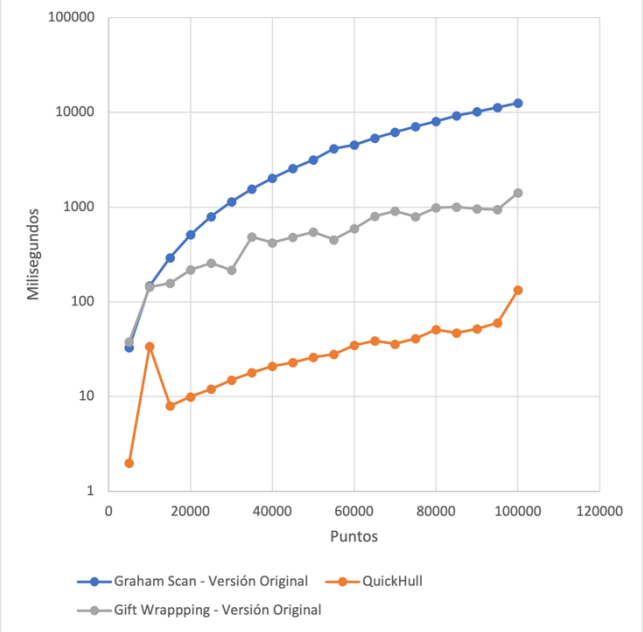


Fig. 13. Comparación del comportamiento entre implementaciones de los algoritmos originales *QuickHull*, *Gift Wrapping* y *Graham Scan*.

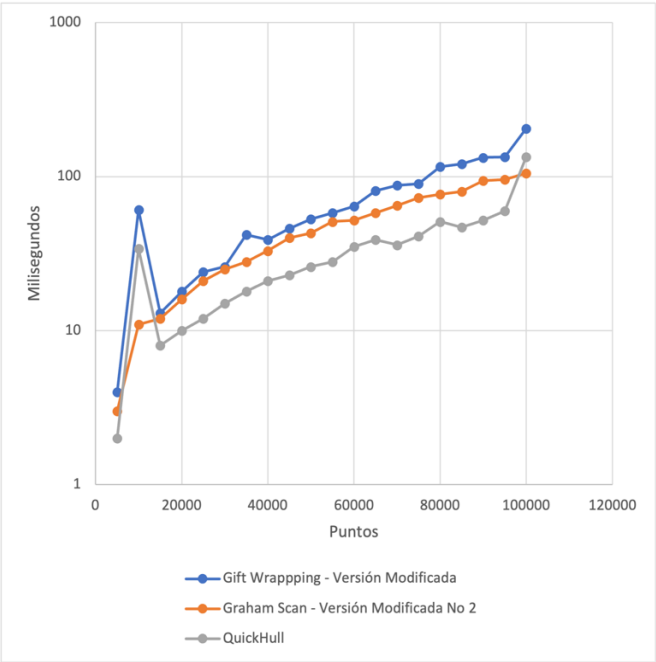


Fig. 14. Comparación del comportamiento entre implementaciones de los algoritmos QuickHull y las versiones modificadas de Gift Wrapping y Graham Scan.

## VI. Conclusiones

Se implementaron los algoritmos *QuickHull*, *Gift Wrapping* y *Graham Scan* para el cálculo del *CH*, usando 20 archivos. Cada archivo incluía una nube de puntos aleatoria. Los archivos generados tenían tamaños que iban desde los 5.000 puntos hasta los 100.000 puntos.

Para los algoritmos *Gift Wrapping* y *Graham Scan* se realizaron implementaciones modificadas con la intención de reducir los tiempos de ejecución. Con respecto a la implementación modificada del algoritmo *Gift Wrapping*, se tomó como estrategia comparar los ángulos formados por los vértices, lo que condujo a significativa reducción en el tiempo de ejecución frente a la versión original de este algoritmo. Para el algoritmo *Graham Scan* se realizaron dos implementaciones modificadas que emplearon como estrategia reducir el tamaño de la nube de puntos.

A partir de la comparación entre las todas las implementaciones realizadas en la investigación, se comprobó que el algoritmo *QuickHull* es el más rápido. También se nota reducción en los tiempos de ejecución en las implementaciones modificadas con relación a las implementaciones originales de los algoritmos *Gift Wrapping* y *Graham Scan*.

Como trabajo futuro, se podrán realizar implementaciones y comparación con otros algoritmos que calculan el *CH* buscando optimización en el tiempo de ejecución. Adicionalmente, se podrá usar el polígono convexo *CH* como una base para encontrar la ruta en la arquitectura de los microservicios que permita la reducir la latencia. También, se podrá realizar la construcción de una herramienta software para apoyar el proceso de enseñanza-aprendizaje del *CH*, permitiendo una fácil evaluación de implementaciones algorítmicas para calcularlo.

## Referencias


[1] S. Bu-Qing and L. Ding-Yuan, Computational geometry: curve and surface modeling. Elsevier, 2014.

[2] M. A. Jayaram and H. Fleyeh, "Convex Hulls in Image Processing: A Scoping Review," American Journal of Intelligent Systems, Artikel PeerReviewed no. 2, p. 48, 2016. [Online].

[3] O. Y. Buitrago, A. L. Ramírez, and R. A. Britto, "Nuevo Algoritmo para la Construcción de la Envolvente Convexa en el Plano," New Algorithm to Construct a Planar Convex Hull., Article vol. 26, no. 4, pp. 137-144, 2015, doi: 10.4067/S0718-07642015000400017.

[4] Y. Xu and W. Hou, "Calculation of operational domain of virtual maintenance based on convex hull algorithm," ed: IEEE, 2017, p. 1.

[5] J. F. Peters, "Foundations of computer vision," ed: Springer International Publishing: 2017.

[6] R. Xu, H. Dai, F. Wang, and Z. Jia, "A convex hull based optimization to reduce the data delivery latency of the mobile elements in wireless sensor networks," in 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013: IEEE, pp. 2245-2252.

[7] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. Toussaint, "Space-efficient planar convex hull algorithms," Theoretical Computer Science, vol. 321, no. 1, pp. 25-40, 2004.

[8] Z. Qihai, "New serial and parallel algorithms for finding convex hull based on clusters, domains and directions from single to multitude," Journal of Algorithms & Computational Technology, vol. 3, no. 2, pp. 191-227, 2009.

[9] H.-Y. CHIANG, "Solving 2D Convex Hull with Parallel Algorithms," 2010.

[10] N.-D. Hoang and N. K. Linh, "Quicker than Quickhull," Vietnam Journal of Mathematics, vol. 43, no. 1, pp. 57-70, 2015, doi: 10.1007/s10013-014-0067-1.

[11] V. Skala, M. Smolik, and Z. Majdisova, "Reducing the number of points on the convex hull calculation using the polar space subdivision in E2," 2016.

[12] A. N. Gamby and J. Katajainen, "Convex-hull algorithms: Implementation, testing, and experimentation," Algorithms, vol. 11, no. 12, p. 195, 2018.
[13] A. N. Gamby and J. Katajainen, "A faster convex-hull algorithm via bucketing," in International Symposium on Experimental Algorithms, 2019: Springer, pp. 473-489.
[14] K. Borna, "Sweep Line Algorithm for Convex Hull Revisited," Journal of Algorithms and Computation, vol. 51, no. 1, pp. 1-14, 2019.
[15] J. Lesjak, "Pregled in primerjava algoritmov za izračun konveksne ovojnice," 2021.
[16] F. P. Preparata and M. I. Shamos, "Computational Geometry An Introduction," in Computational Geometry: Springer, 1985, pp. 1-35.
[17] J.-R. Sack and J. Urrutia, Handbook of computational geometry. Elsevier, 1999.
[18] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, Computational Geometry: Algorithms and Applications, Third Edition ed. Springer, 2008.
[19] J. o'Rourke and A. J. Mallinckrodt, "Computational geometry in C," Computers in Physics, vol. 9, no. 1, pp. 55-55, 1995.
[20] W. F. Eddy, "A new convex hull algorithm for planar sets," ACM Transactions on Mathematical Software (TOMS), vol. 3, no. 4, pp. 398-403, 1977.
[21] A. Bykat, "Convex hull of a finite set of points in two dimensions," Information Processing Letters, vol. 7, no. 6, pp. 296-298, 1978.
[22] N. K. Linh, P. T. An, and T. Van Hoai, "A fast and efficient algorithm for determining the connected orthogonal convex hulls," Applied Mathematics and Computation, vol. 429, p. 127183, 2022.
[23] D. M. Steier and A. P. Anderson, Algorithm Synthesis: A comparative study. Springer Science & Business Media, 2012.
[24] D. R. Chand and S. S. Kapur, "An algorithm for convex polytopes," Journal of the ACM (JACM), vol. 17, no. 1, pp. 78-86, 1970.
[25] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," Info. Pro. Lett., vol. 1, pp. 132-133, 1972.

**Christian Andrés Candela Uribe.**
Estudiante del Doctorado en Ingeniería con énfasis en Ciencias de la Computación de la Universidad Tecnológica de Pereira. Profesor Asociado en la Universidad del Quindío Colombia. Sus áreas de interés son: Arquitectura de Software, Ingeniería de Software y Microservicios.
ORCID: https://orcid.org/0000-0002-3961-1840

**Luis Eduardo Sepúlveda Rodríguez.**
Estudiante del Doctorado en Ingeniería con énfasis en Ciencias de la Computación de la Universidad Tecnológica de Pereira. Profesor Asociado en la Universidad del Quindío Colombia. Sus áreas de interés son: Infraestructura de TI, Sistemas Operativos y Cloud Computing.
ORCID: https://orcid.org/0000-0003-2446-0602

**Julio César Chavarro Porras.**
Doctorado en Ingeniería con énfasis en Ciencias de la Computación de la Universidad del Valle. Profesor Asociado en la Universidad Tecnológica de Pereira Colombia. Sus áreas de interés son: Inteligencia Artificial, Arquitectura de Software, Ingeniería de Software y Bases de Datos.
ORCID: https://orcid.org/0000-0001-8876- 8855

**Carlos Augusto Meneses Escobar.**
Estudiante del Doctorado en Ingeniería con énfasis en Ciencias de la Computación de la Universidad Tecnológica de Pereira. Profesor Asistente en la Universidad Tecnológica de Pereira Colombia. Sus áreas de interés son: Inteligencia Artificial, Arquitectura de Software, Ingeniería de Software y Bases de Datos.
ORCID: https://orcid.org/0000-0002-8192-6889

**John Alexander Sanabria Ordoñez.**
Doctorado en Computer Information Science and Enginnering de la Universidad de Puerto Rico. Profesor Asistente en la Universidad del Valle Colombia. Sus áreas de interés son: Redes de computadores, Sistemas Distribuidos, Grid Computing y Plataformas Computacionales.
ORCID: https://orcid.org/0000-0003-1381-5682

**Olmedo Arcila Guzmán**
Doctorado en Ingeniería con énfasis en Ciencias de la Computación de la Universidad del Valle. Desarrollador de Software. Sus áreas de interés son: Geometría Computacional, Sistemas Operativos y Ingeniería de Software.