



**CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS  
DEL INSTITUTO POLITÉCNICO NACIONAL**

**DEPARTAMENTO DE CONTROL AUTOMÁTICO**

**Versión revisada de un algoritmo  
que determina la cubierta convexa relativa  
de polígonos simples en el plano**

Tesis que presenta

**HUGO REYES BECERRIL**

para obtener el Grado de

**MAESTRO EN CIENCIAS**

en la Especialidad de

**CONTROL AUTOMÁTICO**

Directora de tesis: Dra. Petra Wiederhold Grauert

México, D.F.

Septiembre, 2013



*“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.”*

Sir Charles Antony Richard Hoare  
*British computer scientist (1934-)*



# Agradecimientos

## A MIS PADRES

Por su enorme paciencia, cariño y apoyo incondicional a lo largo de mi vida.

## A LA DRA. PETRA WIEDERHOLD

Por su gran apoyo académico.

## A MIS COMPAÑEROS Y AMIGOS

A Rodrigo Mallén, Román Rodríguez, Miguel Maldonado, Aurora Rodríguez y Manuel Hunter, quienes me hicieron sentir realmente que la escuela era como mi segunda casa.

## A MIS PROFESORES SINODALES

Por el tiempo invertido en la revisión de este trabajo, y por sus valiosos comentarios.

## AL CINVESTAV Y SU DEPARTAMENTO DE CONTROL AUTOMÁTICO

Por permitirme continuar con mi desarrollo académico en una institución de gran prestigio.

## AL CONACyT

Por el apoyo económico, sin el cual me hubiera sido imposible continuar mis estudios.



# Índice general

<b>Resumen</b>	<b>XI</b>
<b>Introducción</b>	<b>XIII</b>
<b>1. Cubierta convexa en el plano</b>	<b>1</b>
1.1. Conceptos básicos . . . . .	1
1.2. Cubierta convexa de un polígono simple . . . . .	5
1.3. Algoritmos . . . . .	8
1.3.1. Concepto de algoritmo . . . . .	8
1.3.2. Estructuras de datos . . . . .	8
1.3.3. Recursión . . . . .	11
1.3.4. Complejidad . . . . .	12
1.4. Aplicaciones de la cubierta convexa . . . . .	13
1.5. Algoritmos que determinan la cubierta convexa . . . . .	14
1.5.1. Generalidades . . . . .	14
1.5.2. Sentidos de giro y convexidad/concavidad local . . . . .	16
1.5.3. Descripción de algunos algoritmos representativos . . . . .	17
1.5.4. Algoritmo de Melkman . . . . .	21
<b>2. Cubierta convexa relativa</b>	<b>23</b>
2.1. Definición y propiedades de la cubierta convexa relativa . . . . .	23
2.2. La cubierta convexa relativa y el polígono de longitud mínima . . . . .	24
2.3. Aplicaciones de la cubierta convexa relativa . . . . .	28
2.4. Algoritmos que determinan la cubierta convexa relativa . . . . .	30
2.5. Algoritmo de G. Klette . . . . .	31
2.5.1. Propiedades importantes . . . . .	31
2.5.2. Descripción detallada . . . . .	32
<b>3. Propuesta de algoritmo</b>	<b>37</b>
3.1. Análisis del algoritmo de G. Klette . . . . .	37
3.1.1. Observaciones . . . . .	37
3.1.2. Fallas en el algoritmo . . . . .	40

3.2. Versión revisada del algoritmo . . . . .	43
3.2.1. Descripción detallada . . . . .	43
3.2.2. Pseudocódigo de la versión revisada . . . . .	50
<b>4. Implementación y Resultados</b>	<b>53</b>
4.1. Justificación del uso de Matlab . . . . .	53
4.2. Implementación del algoritmo de Melkman . . . . .	54
4.3. Implementación de la versión revisada . . . . .	58
4.4. Interfaz gráfica . . . . .	58
4.5. Resultados de la versión revisada . . . . .	59
4.6. Discusión de resultados . . . . .	62
<b>Conclusiones</b>	<b>65</b>
<b>Trabajos futuros</b>	<b>66</b>
<b>Apéndices</b>	<b>68</b>
A. Diagrama de flujo del nuevo algoritmo . . . . .	68
B. Artículo de G. Klette . . . . .	73
C. Artículo de A. Melkman . . . . .	87
<b>Bibliografía</b>	<b>91</b>

# Índice de figuras

1.1. Conjunto convexo (izquierda) y conjunto no convexo (derecha). . . . .	2
1.2. Cubierta convexa de un conjunto. . . . .	3
1.3. Cavidades y hoyos de un conjunto no convexo. . . . .	5
1.4. Polígonos simples (izquierda) y polígonos no simples (derecha). . . . .	7
1.5. La cubierta convexa de un polígono simple (izquierda) es idéntica a la del conjunto de sus vértices (derecha). . . . .	7
1.6. Aplicaciones de la cubierta convexa: Rectángulo envolvente más pequeño alineado con los ejes (izquierda), orientado arbitrariamente (centro), y círculo envolvente más pequeño (derecha) de un conjunto de puntos. . . . .	15
1.7. Trazado de curvas de Jordan y de polígonos en sentido horario. . . . .	16
1.8. Semiplanos generados por la recta orientada desde $p_1$ hacia $p_2$ . . . . .	16
1.9. Tabla comparativa de métodos para determinar la concavidad o convexidad local de una tripleta de puntos. . . . .	18
1.10. Visibilidad desde el exterior: polígono visible (izquierda) y no visible (derecha). Cualquier rayo que parta de $q$ hacia el exterior interseca a $P$ . . . . .	20
1.11. Pseudocódigo del algoritmo de Melkman [7]. . . . .	21
2.1. Cubierta convexa relativa $CH_B(A)$ . . . . .	24
2.2. Retículas con diferente resolución. . . . .	25
2.3. a) Conjunto $S$ y su frontera $\gamma$ , b) digitalización interior $J_h^-(S)$ , c) digitalización exterior $J_h^+(S)$ y d) curva digital $\Gamma$ . . . . .	26
2.4. La cubierta convexa relativa coincide con el $MLP$ . . . . .	28
2.5. Determinación de la longitud de una curva digital. . . . .	29
2.6. Trayectoria de longitud mínima en un plano con restricciones para un robot móvil. . . . .	29
2.7. Cavidades con intersección no vacía y formación de nuevos polígonos. . . . .	32
2.8. Cavidades anidadas. . . . .	34
2.9. Pseudocódigo del algoritmo de G. Klette [2]. . . . .	35
3.1. Resultados del algoritmo de Melkman para entradas en sentido horario (izquierda) y en sentido antihorario (derecha). . . . .	38
3.2. Zonas ciegas en el ejemplo de G. Klette. . . . .	41
3.3. El vértice $p_3$ que forma parte del resultado final no es detectado por el algoritmo. . . . .	42
3.4. Falla del algoritmo de G. Klette en el caso de polígonos isotéticos. . . . .	42
3.5. Casos de colinealidades no contempladas por el algoritmo de G. Klette. . . . .	43
3.6. Polígonos interior y exterior de ejemplo. . . . .	44

3.7. Cubierta principal y cubiertas secundarias. . . . .	46
3.8. Cavidad en $B$ con vértices de $A$ dentro de ella. . . . .	48
3.9. Diferencia entre los polígonos $O_S$ , $O_E$ y $O_{new}$ . . . . .	49
3.10. Pseudocódigo de la versión revisada. . . . .	51
4.1. Error producido por las instrucciones faltantes en el pseudocódigo de [7]. . . . .	55
4.2. Pseudocódigo del algoritmo de Melkman utilizado en la implementación. . . . .	56
4.3. Representación de un polígono en forma matricial. . . . .	57
4.4. Polígonos de ejemplo para prueba comparativa. . . . .	60
4.5. Resultado obtenido aplicando la versión corregida. . . . .	61
4.6. Resultado estimado aplicando el algoritmo de G. Klette. . . . .	61
4.7. Resultados aplicando el algoritmo de G. Klette (izquierda) y la versión corregida (derecha) a polígonos isotéticos. . . . .	62

# Resumen

En esta tesis se presenta una versión revisada de un algoritmo que determina la cubierta convexa relativa de dos polígonos simples en el plano.

El desarrollo de esta versión mejorada surgió después de un exhaustivo análisis del algoritmo recursivo propuesto por G. Klette en la *International Conference on Discrete Geometry for Computer Imagery* (DGCI) 2011, y publicado en [2] y [7]. A partir de este análisis se identificaron diversos casos para los cuales este algoritmo no obtiene los resultados esperados. Estas fallas están relacionadas con la presencia de colinealidades y con la formación de zonas ciegas.

La nueva versión propuesta corrige todas las fallas identificadas en el algoritmo de G. Klette, y para corroborar esta afirmación, fue implementada exitosamente en Matlab, obteniendo resultados satisfactorios.

En el texto se presenta una comparación de resultados utilizando un par de polígonos de prueba diseñado específicamente para evaluar el desempeño de ambos algoritmos en diversas configuraciones típicas, y algunas otras atípicas. En ella se hacen evidentes las fallas del algoritmo analizado, en contraste con el correcto funcionamiento de la versión revisada.

## Abstract

In this thesis a revised version of an algorithm to obtain the relative convex hull of two simple polygons in the plane is introduced.

The new version arises from an exhaustive analysis of the recursive algorithm presented by G. Klette in the *International Conference on Discrete Geometry for Computer Imagery* (DGCI) 2011, and published in [2] and [7]. From this analysis some fail cases were detected, leading to wrong results. Fails are related to colinearities and blind zones.

This improved version corrects all fails previously identified in Klette's algorithm. It was successfully implemented in Matlab to check for proper operation, getting satisfactory results.

A comparative chart is presented to compare the results from the new algorithm and from Klette's algorithm. For this purpose, a pair of test polygons was specifically designed to evaluate the performance of both algorithms. These test polygons include several typical as well as challenging configurations. In the chart, the wrong results of Klette's algorithm and the correct output of the revised version are evident.



# Introducción

## Ubicación y objetivos

Este trabajo se ubica dentro del campo de la **geometría computacional**, la cual estudia algoritmos que resuelven problemas métricos, combinatorios y topológicos de colecciones finitas de objetos geométricos simples, típicamente en el espacio euclidiano.

El **objetivo general** de esta tesis es analizar detalladamente el algoritmo que determina **la cubierta convexa relativa de dos polígonos simples en el plano**, propuesto por G. Klette en la *International Conference on Discrete Geometry for Computer Imagery* (DGCI) celebrada en Francia, en 2011, y verificar los casos en los que se presume que este algoritmo falla.

Los **objetivos particulares** son:

- Estudiar los conceptos de cubierta convexa y cubierta convexa relativa para polígonos simples en el plano, y sus propiedades.
- Analizar los principales algoritmos que determinan la cubierta convexa de un conjunto de puntos y de un polígono simple.
- Implementar el algoritmo de Melkman que determina la cubierta convexa de un polígono simple.
- Estudiar y analizar el algoritmo de G. Klette para determinar la cubierta convexa relativa de dos polígonos simples.
- Verificar algunos casos especiales en los que se piensa que el algoritmo de G. Klette falla.
- Describir las condiciones que generan estos casos de falla, y la forma en que afectan al resultado final.
- Desarrollar una versión corregida del algoritmo de G. Klette o proponer una nueva metodología que no produzca tales errores.
- Implementar la versión corregida, o la nueva metodología del algoritmo, en Matlab.
- Realizar una comparación de resultados entre los obtenidos por el algoritmo de G. Klette y los de la versión corregida.

# Antecedentes

La *cubierta convexa relativa* [5] es una figura en el plano cuya frontera circunscribe a un polígono, llamado polígono interior, y que está restringida por otro polígono llamado polígono exterior. Esta frontera tiene la importante propiedad de ser la curva de longitud más corta entre todas las curvas de Jordan posibles que rodean al polígono interior, y que están dentro del polígono exterior [13].

En general, la cubierta convexa relativa ofrece un método para obtener la trayectoria más corta entre una región anular poligonal. En particular, la cubierta convexa relativa es idéntica (en el plano) al denominado *Polígono de Longitud Mínima (MLP)*, por sus siglas en inglés) [3] [4] [6] [12] [13].

Una de las aplicaciones más importantes de la cubierta convexa relativa se tiene en el campo de imágenes digitales. En esta disciplina existen métodos de estimación de la longitud de una curva a partir de su versión digitalizada, basados en el *MLP*.

Uno de los algoritmos más representativos para determinar la cubierta convexa relativa es el propuesto por Klette-Kovalevsky-Yip [3] en 1999. Sin embargo, este algoritmo está enfocado a la estimación de la longitud de una curva digitalizada, y su uso está limitado a polígonos conformados únicamente por lados horizontales y verticales (isotéticos), y con una separación igual a un píxel.

Otro algoritmo, propuesto por Toussaint [14] en 1986, ofrece un método para determinar la cubierta convexa relativa en el caso general, es decir, para cualquier par de polígonos simples (uno completamente contenido en el otro). Sin embargo, el algoritmo requiere de un complicado proceso de triangulación, el cual no se detalla claramente.

El algoritmo propuesto por G. Klette [2] en 2011, también se postula para el caso general, pero tiene la novedosa característica de ser un algoritmo recursivo. Sin embargo, en este trabajo se muestra que este algoritmo puede obtener resultados erróneos.

## Estructura de la tesis

### Capítulo 1

Se presentan los conceptos básicos acerca de conjuntos convexos y la cubierta convexa en el plano. También se describen algunos de los principales algoritmos para determinar la cubierta convexa. Adicionalmente se detallan algunas de las aplicaciones más comunes de la cubierta convexa en diversas disciplinas.

### Capítulo 2

Se expone la definición de la cubierta convexa relativa, así como algunos algoritmos que han sido propuestos para determinarla. Además, se explica a detalle el algoritmo de G. Klette. También se mencionan algunas aplicaciones de la cubierta convexa relativa.

## Capítulo 3

Se detalla el análisis realizado al algoritmo de G. Klette, describiendo sus fallas y realizando observaciones a su pseudocódigo. Posteriormente, se explica la nueva versión del algoritmo, la cual corrige las fallas encontradas en el algoritmo de G. Klette.

## Capítulo 4

Se expone la forma en que se implementaron tanto el algoritmo de Melkman como la versión revisada del algoritmo de G. Klette que se propone en este trabajo. Después se hace una comparación entre los resultados que se esperarían obtener con el algoritmo de G. Klette, y los obtenidos por la implementación de la versión corregida.

En los apéndices se incluye un diagrama de flujo de la versión implementada, así como los artículos a los que se hace mayor referencia a lo largo del texto, los cuales corresponden al del algoritmo de Melkman [8] para determinar la cubierta convexa, y al de G. Klette [2] para determinar la cubierta convexa relativa.

La versión de software utilizada para implementar los algoritmos fue Matlab R2012a. Por simplicidad, el código y los archivos ejecutables del algoritmo propuesto no se adjuntan al presente documento, sin embargo pueden ser solicitados contactando al autor y/o asesora de este trabajo.

## Contacto

Asesora: Dra. Petra Wiederhold  
E-mail: *[pwiederhold@gmail.com](mailto:pwiederhold@gmail.com)*

Alumno: Hugo Reyes Becerril  
E-mail: *[hrb87@hotmail.com](mailto:hrb87@hotmail.com)*



# Capítulo 1

## Cubierta convexa en el plano

### 1.1. Conceptos básicos

La cubierta convexa es el concepto matemático más importante usado en esta tesis. En esta sección se resumen algunas definiciones y propiedades de conceptos relacionados con la cubierta convexa. Cabe aclarar que en esta tesis se trabaja únicamente en el plano euclidiano  $\mathbb{R}^2$ . Por ello, aunque muchos conceptos discutidos a continuación son conocidos o pueden ser definidos de manera más general para el espacio euclidiano  $\mathbb{R}^n$ , aquí se tratan solamente para el caso  $\mathbb{R}^2$ .

Recordando que  $\mathbb{R}^2$  forma un espacio vectorial con la suma entre vectores y la multiplicación de vectores con escalares, se tiene para  $p = (p_1, p_2)$ ,  $q = (q_1, q_2)$  con  $p_1, p_2, q_1, q_2, \alpha \in \mathbb{R}$  arbitrarios, lo siguiente:

$$(p_1, p_2) + (q_1, q_2) = (p_1 + q_1, p_2 + q_2), \quad \alpha \cdot (p_1, p_2) = (\alpha \cdot p_1, \alpha \cdot p_2).$$

Este espacio vectorial forma un espacio normado con la norma estándar  $\|\cdot\|$ , también llamada norma euclidiana. Esta norma genera la métrica o distancia euclidiana  $d(p, q)$ , con la cual  $\mathbb{R}^2$  forma un espacio métrico. Para  $p = (p_1, p_2), q = (q_1, q_2) \in \mathbb{R}^2$  se tiene entonces que:

$$\|p\| = \sqrt{p_1^2 + p_2^2}, \quad d(p, q) = \|p - q\| = \sqrt{\sum_{i=1}^2 (p_i - q_i)^2}.$$

Usando esta métrica  $d$ , se definen por ejemplo los **discos abiertos**: para  $x \in \mathbb{R}^2$  y  $\epsilon \in \mathbb{R}$ ,  $\epsilon > 0$ ,

$$U_\epsilon(x) = \{y \in \mathbb{R}^2 : d(x, y) < \epsilon\}.$$

Entonces, un subconjunto  $M \subset \mathbb{R}^2$  se llama **acotado** si  $M$  cabe dentro de un disco, es decir, existen  $x \in \mathbb{R}^2$  y  $\epsilon \in \mathbb{R}$ ,  $\epsilon > 0$  tales que  $M \subset U_\epsilon(x)$ .

La cubierta convexa se basa en el concepto de convexidad, el cual se define como sigue:

**Definición 1.1.1** *Un conjunto  $S \subset \mathbb{R}^2$  ( $S \neq \emptyset$ ) se llama **convexo** si para cualquier par de puntos  $p, q \in S$ , el segmento de línea recta  $\overline{pq}$  está completamente contenido en  $S$ .*

Usando las propiedades del espacio vectorial  $\mathbb{R}^2$ , se obtiene que para cualesquiera  $p, q, r \in \mathbb{R}^2$ ,  $r \in \overline{pq}$  si y solo si  $r = p + \lambda(q - p)$  con  $0 \leq \lambda \leq 1$ , lo cual es equivalente a  $r = p + \lambda q - \lambda p =$

$(1 - \lambda)p + \lambda q$ . Tomando en cuenta que tanto  $\lambda$  como  $(1 - \lambda)$  son números entre 0 y 1 y ambos suman 1, se obtienen las siguientes expresiones equivalentes:

$$\begin{aligned} r \in \overline{pq} &\iff r = p + \lambda(q - p) \text{ con } 0 \leq \lambda \leq 1 \\ &\iff r = \lambda q + (1 - \lambda)p \text{ con } 0 \leq \lambda \leq 1 \\ &\iff r = \lambda_1 q + \lambda_2 p \text{ con } \lambda_1, \lambda_2 \in \mathbb{R}, \lambda_1, \lambda_2 \geq 0, \lambda_1 + \lambda_2 = 1. \end{aligned}$$

En consecuencia, una caracterización de la convexidad es la siguiente: para  $S \subset \mathbb{R}^2$ ,

$$S \text{ es convexo} \iff \forall p, q \in S \text{ y } \forall \lambda \in [0, 1] \text{ se sigue que } \lambda q + (1 - \lambda)p \in S.$$

Se puede observar que, por ejemplo, un solo punto o cualquier línea recta forman un conjunto convexo, así como cualquier triángulo, rectángulo o disco. La Figura 1.1 ilustra el concepto de un conjunto convexo.

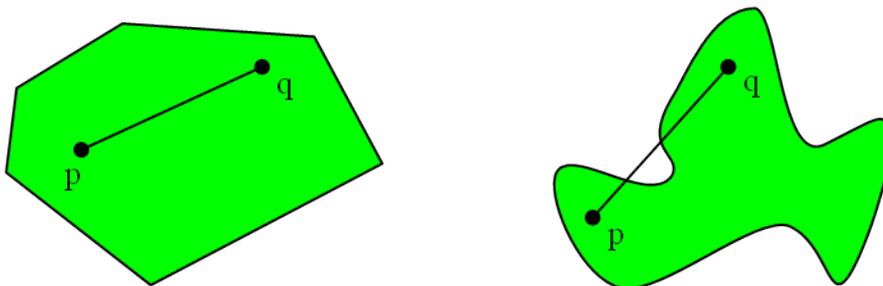


Figura 1.1: Conjunto convexo (izquierda) y conjunto no convexo (derecha).

Es fácil ver que la intersección de un número arbitrario de conjuntos convexos es nuevamente un conjunto convexo. Esto da la motivación para considerar al conjunto convexo mínimo que contiene a un conjunto dado:

**Definición 1.1.2** Para cualquier conjunto  $S \subset \mathbb{R}^2$  no vacío, su cubierta convexa  $CH(S)$  se define como la intersección de todos los conjuntos convexos que contienen a  $S$ .

De esta definición se hace evidente que si un conjunto  $S$  es convexo, entonces  $CH(S) = S$ . Es claro también que la cubierta convexa de un conjunto  $S$  de dos puntos distintos  $S = \{p, q\} \in \mathbb{R}^2$  es el segmento de línea que los une:

$$CH(S) = \overline{pq} = \{x = \lambda_1 p + \lambda_2 q, \lambda_1, \lambda_2 \in \mathbb{R}, \lambda_1, \lambda_2 \geq 0, \lambda_1 + \lambda_2 = 1\}.$$

Más generalmente, se puede demostrar ([1]) que para  $S \subset \mathbb{R}^2$ ,

$$CH(S) = \left\{ \sum_{i=0}^k \lambda_i p_i, p_i \in S, \lambda_i \in \mathbb{R}, \lambda_i \geq 0 \quad \forall i = 0, 1, \dots, k, \sum_{i=0}^k \lambda_i = 1, k \in \mathbb{N} \right\}.$$

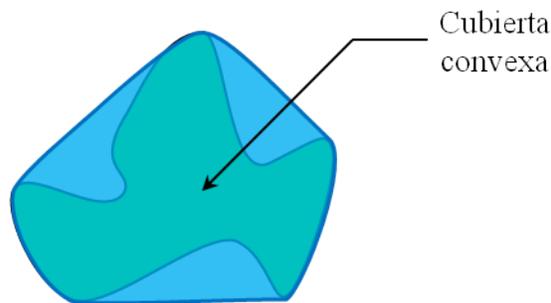


Figura 1.2: Cubierta convexa de un conjunto.

Una combinación lineal  $x = \sum_{i=0}^k \lambda_i p_i$  con  $p_i \in S$ ,  $\lambda_i \in \mathbb{R}$ ,  $\lambda_i \geq 0$  para todo  $i = 0, 1, \dots, k$ ,  $\sum_{i=0}^k \lambda_i = 1$ ,  $k \in \mathbb{N}$ , también es llamada *combinación convexa* de los puntos  $\{p_1, p_2, \dots, p_k\}$ . Un ejemplo de cubierta convexa se expone en la Figura 1.2.

Una representación interesante de cada conjunto convexo se obtiene a partir de semiespacios. Para esto es necesario recordar que cada línea recta en el plano  $\mathbb{R}^2$  es el conjunto de soluciones  $(x, y) \in \mathbb{R}^2$  de una ecuación lineal

$$\alpha_0 + \alpha_1 x + \alpha_2 y = 0,$$

donde los coeficientes son números reales que no cumplen la condición  $\alpha_0 = \alpha_1 = \alpha_2 = 0$ . Dicha línea recta separa al plano en dos regiones, llamadas **semiespacios abiertos** o **semiplanos abiertos**, los cuales se definen como los dos conjuntos de soluciones  $(x, y) \in \mathbb{R}^2$  de las siguientes dos desigualdades:

$$\alpha_0 + \alpha_1 x + \alpha_2 y < 0, \quad \alpha_0 + \alpha_1 x + \alpha_2 y > 0.$$

Cuando se considera cada una de estas dos regiones incluyendo la línea recta, se obtienen los **semiespacios cerrados** o también llamados **semiplanos cerrados**, definidos como los dos conjuntos de soluciones  $(x, y) \in \mathbb{R}^2$  de las siguientes dos desigualdades:

$$\alpha_0 + \alpha_1 x + \alpha_2 y \leq 0, \quad \alpha_0 + \alpha_1 x + \alpha_2 y \geq 0.$$

Es evidente que cada semiespacio, cerrado o abierto, es un conjunto convexo. Más interesante es que cualquier intersección de semiespacios (cerrados o abiertos), cuando no es vacía, es un conjunto convexo. Pero también es válido lo contrario, es decir, es válida la siguiente propiedad importante:

**Lema 1.1.3** *Un subconjunto de  $S \subset \mathbb{R}^2$  es convexo y cerrado, si y solo si  $S$  es una intersección de semiespacios cerrados.*

Hay que recordar que un subconjunto  $S \subset \mathbb{R}^2$  es cerrado cuando contiene su frontera. Formalmente, estos conceptos se refieren al espacio topológico euclidiano  $\mathbb{R}^2$ , cuya topología  $\tau$  es la euclidiana dada por la base de los discos abiertos  $U_\epsilon(x)$ , los cuales ya fueron definidos anteriormente. Así, cada abierto (es decir, cada elemento de  $\tau$ ) es una unión de discos abiertos,

y un subconjunto de  $\mathbb{R}^2$  es cerrado si su complemento es abierto. Siguiendo los estándares de la literatura [9], se define y se denota para  $M \subset \mathbb{R}^2$ ,

$$\begin{aligned} cl(M) &= \cap \{A \subset \mathbb{R}^2 : M \subseteq A, A \text{ cerrado}\} && \text{(cerradura)} \\ int(M) &= \cup \{A \subset \mathbb{R}^2 : A \subseteq M, A \text{ abierto}\} && \text{(interior)} \\ fr(M) &= cl(M) \setminus int(M) && \text{(frontera).} \end{aligned}$$

También son necesarios los conceptos de componente conexa y de curva. Una **curva** está definida como la imagen del intervalo cerrado  $[0, 1]$  bajo una función continua  $f : \mathbb{R} \rightarrow \mathbb{R}^2$ , es decir, una curva es el conjunto  $\{f(x) : x \in [0, 1]\}$ . La función  $f$  también es llamada **trayectoria** o *camino*.

En general, un espacio topológico se llama **conexo** si no es representable como unión de dos subconjuntos propios no vacíos abiertos y disjuntos. En esta tesis donde se trabaja solamente en el espacio topológico euclidiano  $\mathbb{R}^2$ , el concepto más estricto de la arco-conexidad es de mayor interés.

**Definición 1.1.4** *Un subconjunto  $M \subset \mathbb{R}^2$  no vacío se llama **arco-conexo** si para cualesquiera dos puntos  $x, y$  de  $M$  existe una curva que une a estos dos puntos, y que además está completamente contenida en  $M$ .*

La arco-conexidad implica la conexidad, pero en general la implicación contraria no es válida [9]. Una consecuencia inmediata de la definición es que cualquier conjunto convexo en  $\mathbb{R}^2$  es también arco-conexo y por lo tanto conexo.

Para  $M \subset \mathbb{R}^2$  no vacío, una **componente conexa o de conexidad**  $C$  de  $S$  es cualquier subconjunto conexo máximo de  $S$ . Esto significa que siempre cuando  $N$  es un subconjunto conexo de  $M$  el cual contiene a  $C$ , entonces  $N = C$ .

Un conjunto no convexo se distingue de su cubierta convexa por la presencia de hoyos y/o cavidades, los cuales se definen a continuación:

**Definición 1.1.5** *Para  $S \subseteq \mathbb{R}^2$ ,  $S \neq \emptyset$ , cualquier componente conexa acotada del conjunto  $S^c = (\mathbb{R}^2 \setminus S)$  se llama **hoyo** de  $S$ .*

**Definición 1.1.6** *Para  $S \subseteq \mathbb{R}^2$ ,  $S \neq \emptyset$ , la cerradura topológica de cualquier componente conexa de  $(CH(S) \setminus S)$  se llama **cavidad** de  $S$ , y es denotada por  $CAV_i(S)$ , donde  $i$  es un índice.*

Obsérvese que para un conjunto cerrado  $S$ , la frontera de un hoyo de  $S$  está completamente contenida en  $S$ , a diferencia de una cavidad, cuya frontera no está completamente contenida en  $S$ .

En la Figura 1.3 se muestran ejemplos de hoyos y cavidades de conjuntos no convexos. De la figura es claro que cada cavidad de  $S$  tiene en su frontera un segmento de línea recta el cual pertenece a la (frontera de la) cubierta convexa  $CH(S)$  pero no es parte de la frontera de  $S$ . Estos segmentos son de gran importancia en este trabajo. Por ello a continuación se les definirá:

**Definición 1.1.7** *Sea  $S \subseteq \mathbb{R}^2$ ,  $S \neq \emptyset$  un conjunto no convexo. Todo segmento de línea recta de longitud máxima que pertenece a la frontera de exactamente una cavidad de  $S$ , pero que no es parte de la frontera de  $S$ , es llamado una **cubierta** de  $S$ .*

El requerimiento de máxima longitud se refiere a que para toda cavidad de  $S$ , entre todos los segmentos de línea recta que pertenece a la frontera de esta cavidad pero que no pertenecen a la frontera de  $S$ , la cubierta es elegida como el segmento más largo. Para cada cavidad, su cubierta está definida de forma única. Resulta entonces que cada cubierta es parte de la frontera de  $CH(S)$ , y separa el interior de exactamente una cavidad, del exterior de  $CH(S)$  (Figura 1.3).

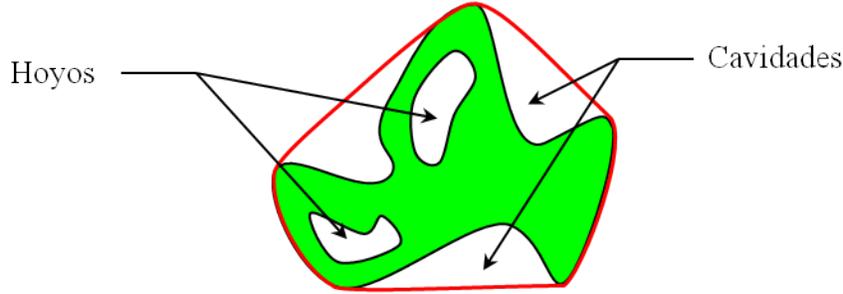


Figura 1.3: Cavidades y hoyos de un conjunto no convexo.

## 1.2. Cubierta convexa de un polígono simple

En esta tesis es de gran interés determinar la cubierta convexa no de subconjuntos arbitrarios del plano euclidiano  $\mathbb{R}^2$ , sino de polígonos simples.

Intuitivamente, se tiene el concepto de un **polígono** como una figura determinada por una secuencia finita de al menos tres puntos  $\{p_1, p_2, \dots, p_k\}$  en el plano donde cada dos puntos consecutivos forman un segmento de línea recta  $\overline{p_i p_{i+1}}$  (para  $i = 1, 2, \dots, k$ , con  $p_{k+1} = p_1$ ). El polígono determinado por  $\{p_1, p_2, \dots, p_k\}$  es considerado como el subconjunto del plano el cual es topológicamente cerrado y acotado, y está encerrado por la curva descrita por la secuencia de segmentos de línea recta:

$$\{ \overline{p_1 p_2}, \overline{p_2 p_3}, \dots, \overline{p_{k-1} p_k}, \overline{p_k p_1} \} .$$

Para formalizar la definición de un polígono simple, concepto que es de suma importante en esta tesis, se especificarán primero algunos tipos de curva.

De acuerdo a la sección anterior, una curva  $\gamma$  es generada a partir de una función continua (trayectoria)  $f : \mathbb{R} \rightarrow \mathbb{R}^2$  definida sobre el intervalo cerrado  $[0, 1]$ :

$$\gamma = \{ f(s) = (x(s), y(s)) \in \mathbb{R}^2 : s \in [0, 1] \} .$$

Dicha curva  $\gamma$  se llama

- *curva abierta* si  $f(0) \neq f(1)$ .
- *curva cerrada* si  $f(0) = f(1)$ .
- **curva simple** si para cualesquiera  $s, t \in [0, 1]$  distintos tales que  $0 \leq s < t < 1$ , se sigue que  $f(s) \neq f(t)$ .

- **curva de Jordan** si es una curva cerrada y simple.
- **arco de Jordan** si es una curva abierta y simple.
- **polilínea** si existe una secuencia finita de puntos  $\{s_0, s_1, s_2, \dots, s_k\}$  con  $0 = s_0 < s_1 < s_2 < \dots < s_k = 1$  tales que todos los segmentos de curva  $\{f(s) : s_i \leq s \leq s_{i+1}\}$  (para  $i = 0, 1, \dots, k-1$ ) son segmentos de línea recta. Estos segmentos  $\overline{p_i p_{i+1}}$  ( $p_i = f(s_i)$ ,  $p_{i+1} = f(s_{i+1})$ ) se llaman los **lados** de la polilínea.
- **polilínea cerrada** si es una polilínea que corresponde a una curva cerrada.
- **polilínea simple** si es una polilínea que corresponde a una curva simple.

Una curva de Jordan intuitivamente cumple con el hecho de que esta curva no se toca ni se cruza a sí misma. Según el teorema de Jordan [9], una curva de Jordan  $\gamma$  separa al plano euclidiano  $\mathbb{R}^2$  en dos regiones unívocamente definidas: una es el conjunto acotado y encerrado por la curva  $\gamma$ , y se llama **interior de la curva de Jordan**  $\gamma$ , la otra región es no acotada y se llama **exterior de la curva de Jordan**  $\gamma$ . Ambas regiones son subconjuntos abiertos del plano (no contienen a la curva  $\gamma$ ) y las dos regiones son disjuntas.

Una polilínea comúnmente es representada solo por sus vértices  $p_0 = f(s_0)$ ,  $p_1 = f(s_1)$ ,  $\dots$ ,  $p_k = f(s_k)$ . Nótese que  $p_0 = f(0) = f(1) = p_k$  para el caso de una polilínea cerrada. En consecuencia, una polilínea simple y cerrada corresponde a una curva de Jordan la cual puede ser representada por la secuencia finita de sus vértices:

$$\{p_1, \dots, p_k\} \quad \text{o} \quad \langle p_0, \dots, p_{k-1} \rangle, \quad \text{donde} \quad p_k = p_0, \quad p_{k+1} = p_1, \quad \text{es decir, es cíclica}$$

o por la secuencia de sus lados:

$$\{\overline{p_1 p_2}, \overline{p_2 p_3}, \dots, \overline{p_{k-1} p_k}, \overline{p_k p_1}\}.$$

**Definición 1.2.1** *Un **polígono simple** se define como cualquier subconjunto no vacío  $P \subset \mathbb{R}^2$  el cual es (topológicamente) cerrado y acotado, y cuya frontera forma una polilínea simple y cerrada.*

En consecuencia, la frontera de un polígono simple es una curva de Jordan que tiene una representación natural mediante la secuencia finita de sus vértices  $\{p_1, \dots, p_k\}$  o, equivalentemente, mediante la secuencia de sus lados  $\{\overline{p_1 p_2}, \dots, \overline{p_{k-1} p_k}, \overline{p_k p_1}\}$  (Figura 1.4).

Un tipo especial de polígono simple es un **polígono convexo**. Este tipo de polígonos siempre puede ser representado como una intersección finita de semiespacios cerrados de  $\mathbb{R}^2$ . Cuando un polígono simple no es convexo, entonces se distingue de su cubierta convexa por tener al menos una cavidad. A su vez, por definición, un polígono simple no puede presentar hoyos.

Una observación sencilla pero importante es la siguiente (Figura 1.5):

**Lema 1.2.2** *La cubierta convexa de un polígono simple es igual a la cubierta convexa del conjunto de sus vértices.*

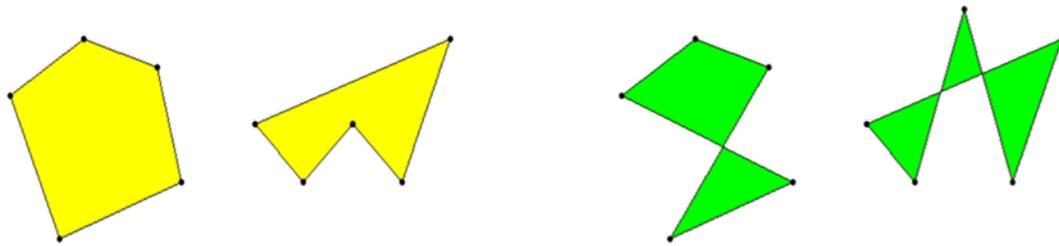


Figura 1.4: Polígonos simples (izquierda) y polígonos no simples (derecha).

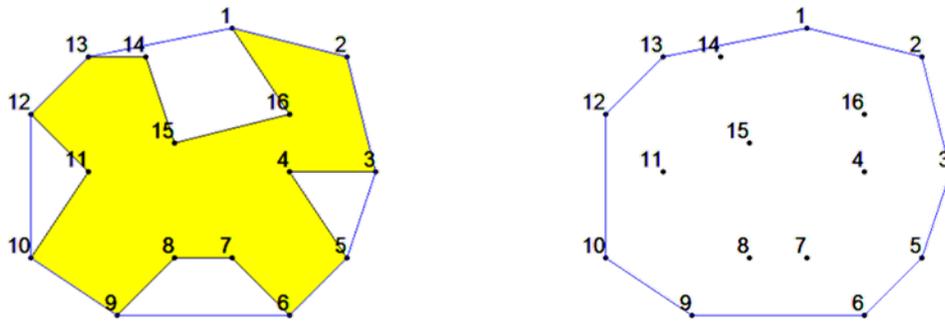


Figura 1.5: La cubierta convexa de un polígono simple (izquierda) es idéntica a la del conjunto de sus vértices (derecha).

La importancia de esta observación radica en el hecho de que un polígono simple tiene un número finito de vértices, lo cual justifica la determinación de la cubierta convexa de un polígono simple mediante algoritmos. Para el desarrollo de tales algoritmos, las siguientes propiedades de la cubierta convexa pueden ser de gran ayuda:

**Lema 1.2.3** Sean  $P$  un polígono simple y  $CH(P)$  su cubierta convexa. Entonces:

1.  $CH(P)$  es el polígono convexo de menor área que contiene a  $P$  [11, p. 65].
2.  $CH(P)$  es el polígono convexo de perímetro más corto que contiene a  $P$  [11, p. 65].

## 1.3. Algoritmos

Puesto que esta tesis implica el análisis, el desarrollo y la implementación de algoritmos, en esta sección se discuten brevemente algunos conceptos y propiedades de algoritmos.

### 1.3.1. Concepto de algoritmo

De acuerdo a [17], un **algoritmo** es un conjunto ordenado y finito de pasos que permite hallar la solución de un problema. Esta idea ya había sido planteada desde los antiguos griegos, sin embargo, las capacidades tecnológicas actuales le han dado una nueva perspectiva a este concepto al resolver problemas utilizando sistemas computacionales.

Desde el punto de vista computacional, se dice que un algoritmo realiza una tarea o resuelve un problema mediante el procesamiento de datos de entrada, arrojando como resultado datos de salida. Este procesamiento se realiza ejecutando operaciones básicas en forma secuencial, las cuales pueden ser expresadas en forma verbal explicativa (e.g. en pseudocódigo), o mediante algún lenguaje de programación.

### 1.3.2. Estructuras de datos

Al referirse a una *estructura de datos* se entiende una forma de organizar datos, que pueden ser tanto datos de entrada como también datos de salida de un algoritmo. En cuanto al concepto de datos, en un sentido amplio, implica cualquier tipo de información. En el contexto de esta tesis, los *datos* son secuencias de puntos en el plano, dados por sus coordenadas. Una estructura de datos pretende organizar información de una manera útil para facilitar el desarrollo y la ejecución de un algoritmo. Una estructura de datos también debe facilitar el proceso de cómo los datos de entrada son transformados, paso a paso, hacia los datos de salida durante la ejecución de un algoritmo. De hecho, la selección o la operación de una estructura de datos se relaciona íntimamente con el desarrollo del algoritmo, así como con el entendimiento de la tarea a realizar o del problema a resolver.

El desarrollo de la programación ha llevado a la utilización generalizada de algunas formas de organización de datos, es decir, de algunas estructuras de datos ya bien conocidas y con propiedades bien definidas.

Las estructuras de datos más comunes en el ámbito de algoritmos y programación son conocidas como listas, pilas y colas, las cuales son descritas en esta sección. La explicación de cada

una de ellas permitirá un mejor entendimiento de la estructura de datos en la que se basan los algoritmos descritos y analizados en esta tesis, denominada *cola doble* o bidireccional. Para cada estructura se explicará inicialmente su forma de organización, y después las operaciones que permiten modificarlas.

Las listas, pilas y colas son estructuras lineales de datos, las cuales son modificadas al agregar o eliminar elementos. Esto tiene como consecuencia inmediata que el número de elementos cambia en dependencia de la operación efectuada. Por eso se dice en el contexto de las ciencias de la computación que estas estructuras son dinámicas.

Una **lista** (*list*, en inglés) es una secuencia finita y ordenada de elementos

$$L = (x_1, x_2, x_3, \dots, x_k), \quad k \geq 1,$$

donde al número de elementos de  $L$  se le denomina la *longitud de la lista*. En esta tesis se trabaja con listas cuyos elementos representan puntos en el plano (donde cada punto está dado por dos coordenadas). En algunos lenguajes de programación, una lista es interpretada como un vector columna. Una lista puede ser modificada al agregar o eliminar un elemento de cualquier posición en ella. Solamente es posible agregar o eliminar un elemento a la vez de la lista. El hecho de que un elemento pueda ser agregado o eliminado en cualquier posición, proporciona mucha flexibilidad al programador, pero internamente requiere mucho trabajo de organización. Se dice también que se puede acceder a cualquier elemento o posición dentro de la lista.

Las operaciones básicas para una lista son:

**Insertar** (*INSERT*): Agrega un elemento nuevo  $m$  en una posición (arbitraria)  $i$  indicada. Esto significa que la operación usa la posición  $i$  como parámetro, donde  $i$  es cualquier número entre 1 y  $k$ . La longitud de la lista aumenta en 1.

$$INSERT(L, i, m) = (x_1, \dots, x_{i-1}, m, x_i, x_{i+1}, \dots, x_k)$$

**Eliminar** (*DELETE*): Borra el  $i$ -ésimo elemento de la lista ( $i$  arbitrario con  $1 \leq i \leq k$ ). La longitud de la lista disminuye en 1.

$$DELETE(L, i) = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$$

Una **pila** (*stack*, en inglés) es una lista  $L = (x_1, x_2, \dots, x_k)$  cuyo acceso es limitado a solamente el último elemento  $x_k$ . Esto significa una restricción considerable al programador, pero a la vez simplifica enormemente la organización interna requerida. Ahora, agregar o eliminar un elemento solamente se realiza al final de la lista, lo cual hace innecesario el uso de algún parámetro de posición. Algunos lenguajes de programación se basan en la idea intuitiva de que una pila es un vector vertical que tiene su primer elemento  $x_1$  abajo (*bottom*) y su último elemento arriba (*top*). Por eso, agregar un elemento significa aumentar la lista por arriba (de hecho, en su final) con un nuevo elemento. Esto requiere ajustar el índice del último elemento sumándole uno. En cambio, eliminar un elemento significa eliminar al último elemento  $x_k$ , y por tanto disminuir el último índice por uno. Debido a esta forma de manipulación, una pila es

conocida como una estructura de *últimas entradas, primeras salidas* (*LIFO* por *last-in/first-out*, en inglés).

Las operaciones básicas para una pila son:

**Insertar al final** (*PUSH*): Agrega un nuevo elemento  $m$  al final de la pila.

$$PUSH(L, m) = (x_1, \dots, x_k, m)$$

**Eliminar al final** (*POP*): Borra el último elemento de la pila.

$$POP(L) = (x_1, \dots, x_{k-1})$$

Una **cola** (*queue*, en inglés) es una lista  $L = (x_1, x_2, \dots, x_k)$  cuyo acceso es limitado de tal manera que sus elementos solo pueden ser eliminados por el extremo inicial de la lista, y los nuevos elementos pueden agregarse únicamente por su extremo final. Intuitivamente esto significa que un elemento puede retirarse (primero) de la lista siempre cuando haya sido introducido primero. Por ello, una cola es conocida como una estructura de *primeras entradas, primeras salidas* (*FIFO* por *first-in/first-out*, en inglés).

Las operaciones básicas para una cola son:

**Insertar al final** (*ENQUEUE*): Agrega un nuevo elemento  $m$  al final de la cola.

$$ENQUEUE(L, m) = (x_1, \dots, x_k, m)$$

**Eliminar al inicio** (*DEQUEUE*): Borra el primer elemento de la cola.

$$DEQUEUE(L) = (x_2, x_3, \dots, x_k)$$

La estructura de datos más relevante para el análisis de los algoritmos estudiados, y la más importante dentro de esta tesis es la **cola doble** o *cola bidireccional* (en inglés **deque** por *double-ended queue*). Esta es una lista  $L = (x_1, x_2, \dots, x_k)$  en la cual los elementos pueden ser agregados o eliminados solo por sus (ambos) extremos. De forma similar a una pila, una cola doble puede ser interpretada como vector columna, donde el extremo inicial es el inferior (*bottom*) y el extremo final es el superior (*top*). Por esto, se usan las operaciones referentes al extremo final (arriba) análogas a las pilas (*PUSH*, *POP*). Para el extremo opuesto se utilizan las operaciones *INSERT* y *DELETE*, pero restringidas a operar únicamente en el primer elemento.

Las operaciones básicas para una *deque* son:

**Insertar al final** (*PUSH*): Agrega un nuevo elemento  $m$  al final de la *deque*.

$$PUSH(L, m) = (x_1, \dots, x_k, m)$$

**Eliminar al final** (*POP*): Borra el último elemento de la *deque*.

$$POP(L) = (x_1, \dots, x_{k-1})$$

**Insertar al inicio** (*INSERT*): Agrega un nuevo elemento  $m$  al inicio de la *deque*.

$$INSERT(L, m) = (m, x_1, \dots, x_k)$$

**Eliminar al inicio** (*DELETE*): Borra el primer elemento de la *deque*.

$$DELETE(L) = (x_2, x_3, \dots, x_k)$$

Cabe recalcar que, en lo posterior, a esta estructura se le denominará solo como *deque*, por ser un tecnicismo ampliamente aceptado dentro de las disciplinas computacionales.

### 1.3.3. Recursión

La característica más importante del algoritmo estudiado en esta tesis, es el uso de la recursión. Intuitivamente, una **función recursiva** es una función que se llama a sí misma, ya sea de forma directa, o indirecta a través de otra función. Su uso se justifica en el tratamiento de problemas cuya solución puede obtenerse resolviendo instancias más pequeñas del mismo problema.

De forma general, un problema que puede ser resuelto mediante recursión se puede expresar en dos partes: la base y la recursión. Para ejemplificar esto, se tomará el problema típico del cálculo del factorial de un número  $n$ , el cual puede expresarse como:

$$0! = 1, \quad n! = n \cdot (n - 1)!, \quad \forall n \in \mathbb{N}$$

Aquí, la expresión de la izquierda constituye **la base**, que no puede descomponerse en casos más simples, y su resultado es bien conocido, trivial o fácil de resolver. En contraste, la expresión de la derecha es **la recursión**, que indica cómo el problema principal puede descomponerse. Sin embargo, esta descomposición no puede ser arbitraria, sino que debe realizarse de tal forma que las instancias resultantes queden expresadas ya sea en términos de la base o de una nueva recursión, pero más simple o más pequeño que el problema original.

Cuando se llega a un nuevo caso recursivo, se llama nuevamente a la función recursiva para descomponer otra vez el problema resultante en otro aún más simple.

A su vez, cuando se llega a la base, la función recursiva resuelve el problema y envía el resultado obtenido a la instancia previa. En ella se opera el resultado con procedimientos sencillos derivados de la descomposición, y el nuevo resultado se envía nuevamente a la instancia previa, y así sucesivamente hasta que se llega a la instancia inicial donde se obtiene el resultado final del problema principal. A cada ocasión en que se llama a la función recursiva se le llama *paso recursivo*.

Es importante resaltar que el éxito en la utilización de una función recursiva dentro de un algoritmo depende de dos factores. Uno es que el problema pueda descomponerse en instancias más pequeñas pero de la misma estructura que el problema original. El otro es que se asegure que siempre se llegue a un caso base, sin importar el número de pasos recursivos que sean necesarios, mientras sea finito. De no cumplirse estas condiciones, la función recursiva operará de forma indefinida o concluirá con un resultado incorrecto.

### 1.3.4. Complejidad

El tiempo requerido para ejecutar un algoritmo en una computadora depende de diversos parámetros tales como el espacio de memoria disponible o el tiempo de ejecución de cada operación aplicada. Sin embargo, no tiene significado alguno expresar el tiempo requerido en un valor absoluto en nano-, micro- o milisegundos, ya que las computadoras frecuentemente difieren en sus características. Por ello, las mediciones para el tiempo de cómputo necesitan ser independientes de la configuración de las computadoras para expresar la calidad de un algoritmo.

Un enfoque común en ciencias de la computación es el uso de una medida abstracta para estimar el tiempo de ejecución de un algoritmo llamada *complejidad en tiempo* o *complejidad computacional*. Intuitivamente, la complejidad en tiempo de un algoritmo describe el número de operaciones básicas que necesitan ser efectuadas para que el algoritmo encuentre la solución a un problema dado, en función del número de datos básicos de entrada. Se consideran como operaciones básicas de cómputo a las operaciones aritméticas, las pruebas lógicas y las instrucciones de control. En general, la idea es asignar a cada operación básica una *unidad de tiempo* y luego sumar todas las unidades de tiempo utilizadas para efectuar al algoritmo. Tal unidad de tiempo no es medida en una escala física, sino que se trata de una unidad abstracta por cada operación básica. Con este enfoque, la estimación del tiempo de ejecución de un algoritmo es independiente del progreso en las tecnologías de cómputo.

Es claro que desarrollar algoritmos de baja complejidad en tiempo o reducir la complejidad computacional de un algoritmo conocido, son metas importantes de interés tanto práctico como también teórico.

En general, la complejidad en tiempo depende del tamaño del conjunto de datos de entrada que tienen que ser procesados por el algoritmo. Por eso, esta complejidad matemáticamente es expresada en dependencia de la cantidad de datos de entrada. Si los datos de entrada son, por ejemplo,  $n$  puntos, entonces la complejidad es expresada como una función  $f(n)$ .

Estas ideas se formalizan matemáticamente como sigue:

**Definición 1.3.1** Sean  $f$  y  $g$  funciones del conjunto de números naturales  $\mathbb{N}$  al conjunto de reales positivos  $\mathbb{R}^+$ , y  $n \geq 0$  un parámetro que indica el tamaño del conjunto de datos de entrada.

- Se dice que  $f$  está acotado por arriba por  $g$ , si existen  $n_0, c \in \mathbb{R}^+$  tales que para todo  $n \geq n_0$  se sigue que  $f(n) \leq c \cdot g(n)$ . Se escribe entonces  $f(n) = \mathcal{O}(g(n))$ .
- Se dice que  $f$  está acotado por debajo por  $g$ , si existen  $n_0, d \in \mathbb{R}^+$  tales que para todo  $n \geq n_0$  se sigue que  $d \cdot g(n) \leq f(n)$ . Se escribe entonces  $f(n) = \Omega(g(n))$ .

En general, se utiliza la cota superior  $\mathcal{O}(g(n))$  para caracterizar el tiempo de ejecución de un algoritmo. Sin embargo, algunos otros conceptos relacionados con estas cotas también serán utilizados en el texto, por lo cual serán descritos a continuación.

Se denomina *complejidad en tiempo en el peor caso* al mayor tiempo de ejecución posible que puede tardar un algoritmo en finalizar, dado cualquier dato de entrada de tamaño  $n$ . Este caso queda expresado por la cota superior. A su vez, la *complejidad promedio o esperada* se refiere al tiempo de ejecución esperado de un algoritmo para cualquier dato de entrada aleatorio de tamaño  $n$ .

Una disciplina matemática que ha contribuido mucho al desarrollo de herramientas para describir la complejidad de algoritmos, es la geometría computacional. Sus objetos de estudio son colecciones finitas de objetos geométricos en el espacio euclidiano. Esta disciplina estudia y pretende desarrollar algoritmos eficientes para resolver problemas acerca de tales colecciones de objetos, y tiene un interés particular en analizar la complejidad de tales algoritmos cuando el número de objetos se incrementa.

La determinación de la cubierta convexa de un polígono simple es un ejemplo de un problema estudiado por la geometría computacional, el cual es de particular interés en esta tesis. Si  $P$  es un polígono simple, la cubierta convexa  $CH(P)$  es un polígono cuyo conjunto de vértices es un subconjunto del conjunto de los vértices del polígono original  $P$ .

Como se verá más tarde, los mejores algoritmos conocidos para determinar la cubierta convexa de un conjunto de  $n$  puntos en el plano tiene una complejidad en tiempo del orden  $\mathcal{O}(n \cdot \log n)$ . En este caso, es importante mencionar que los  $n$  puntos dados como datos de entrada no se asumen como ordenados. De hecho, el término  $\log n$  dentro de la complejidad se origina por el ordenamiento de los puntos que debe ser efectuado como parte de estos algoritmos. En cambio, para el caso en que estos  $n$  puntos de entrada son los vértices de un polígono simple  $P$ , y son dados como una secuencia ordenada  $\{p_1, p_2, \dots, p_n\}$ , entonces hay algoritmos con una complejidad en tiempo de  $\mathcal{O}(n)$  para determinar la cubierta convexa de  $P$ .

## 1.4. Aplicaciones de la cubierta convexa

Algunas de las disciplinas que más se han beneficiado de los avances en algoritmos que determinan la cubierta convexa han sido la de gráficos por computadora, procesamiento de imágenes y reconocimiento de patrones. Sin embargo, los beneficios no se han limitado a ramas de ciencias de la computación, al dar alternativas de solución a problemas de disciplinas tales como la robótica y la geografía.

Cabe destacar que el cálculo de la cubierta convexa generalmente solo es una parte de otros algoritmos que resuelven otros problemas de la geometría. Es decir, en diversas aplicaciones un algoritmo que determina la cubierta convexa es implementado únicamente como un módulo que antecede a otros módulos, y todos ellos en conjunto forman un algoritmo que resuelve un problema a mayor escala.

Algunas aplicaciones de la cubierta convexa son las siguientes:

- **Evasión de colisiones.**- Si la cubierta convexa de un robot evita colisiones con obstáculos, el robot mismo también lo hará. Tomando en cuenta que el robot puede tener una forma no tan fácilmente modelable, la planeación de una trayectoria (aunque sea preliminar) libre de colisiones es más sencilla utilizando la cubierta convexa de un robot [11].
- **Intersección entre entidades gráficas.**- Un recuadro envolvente (*bounding box*) es un descriptor básico que se utiliza ampliamente en tareas de visualización y gráficos por computadora para aproximar la ubicación y magnitud de un objeto de forma irregular, dentro de un entorno gráfico bidimensional (o tridimensional). Algunas aplicaciones requieren detectar intersecciones entre dos o más objetos dentro de este tipo de entornos. Generalmente, para no comprometer los recursos de cómputo, únicamente se comprueba

la intersección entre los recuadros envolventes de los objetos a analizar, los cuales pueden ser de formas intrincadas. Sin embargo esta técnica puede no obtener resultados correctos puesto que el área que comprende el recuadro envolvente puede ser bastante mayor a la del objeto mismo (dependiendo de su forma), y por consiguiente, la probabilidad de que la intersección de dos recuadros envolventes implique la intersección de los objetos a analizar no es tan alta. Una alternativa a este problema es utilizar la cubierta convexa del objeto como su descriptor de ubicación y magnitud. Aunque hacer esto requiere de mayor procesamiento, la cubierta convexa tiene una forma más similar a la del objeto y, en general, es más fácil obtener resultados más precisos de acuerdo al argumento previo [11].

- **Reconocimiento de patrones.-** Un objeto de interés (por ejemplo, presente en una imagen digital) puede ser reconocido en base a sus características de tamaño y/o de forma, mediante la técnica de apareamiento (*matching*). En particular, cuando se usan características geométricas relacionadas con la convexidad y el estudio de concavidades, se aplican estructuras como los “árboles de deficiencia convexa” en el proceso de identificación y clasificación. Estas estructuras se basan en la determinación de la cubierta convexa y sus propiedades [11].
- **Diámetro de conjuntos.-** El diámetro de un conjunto de puntos  $M = \{p_1, p_2, \dots, p_n\}$  dentro de un espacio métrico  $(X, d)$  se define como la mayor distancia posible entre dos puntos cualesquiera del conjunto:  $\text{diam}(M) = \max\{d(p, q) : p, q \in M\}$ . Este diámetro es igual a la distancia entre dos vértices particulares de la cubierta convexa del conjunto [11]. La ventaja de calcular  $\text{diam}(M)$  utilizando solo los vértices de  $CH(M)$  radica en el hecho de que generalmente  $CH(M)$  tiene un número de vértices mucho menor que  $n$ . El cálculo de  $\text{diam}(M)$  también es útil en la obtención del círculo más pequeño que contiene a  $M$  (*smallest enclosing circle*) el cual es un problema de la geometría (Figura 1.6).
- **El rectángulo más pequeño.-** Un problema típico de la geometría es encontrar el rectángulo más pequeño que contiene a un polígono o a un conjunto de puntos (*minimum bounding box*). Estos rectángulos pueden ser de dos tipos: alineados con los ejes cartesianos, u orientado arbitrariamente. Su relación con la cubierta convexa radica en que al menos un lado del rectángulo buscado pasa sobre al menos un lado de la cubierta convexa. Por ello, como primer paso de los algoritmos del rectángulo envolvente más pequeño es el cálculo de la cubierta convexa [16] (Figura 1.6).
- **Cubierta convexa relativa.-** Como se describirá a detalle en el Capítulo 2, el algoritmo para determinar la cubierta convexa relativa que se estudia en esta tesis requiere del cálculo de la cubierta convexa en distintas etapas del procedimiento principal [2] [7].

## 1.5. Algoritmos que determinan la cubierta convexa

### 1.5.1. Generalidades

La cubierta convexa  $CH(S)$  de cualquier conjunto finito  $S$  de puntos en el plano es un polígono simple cuya frontera es una polilínea simple cerrada. La mayoría de los algoritmos que

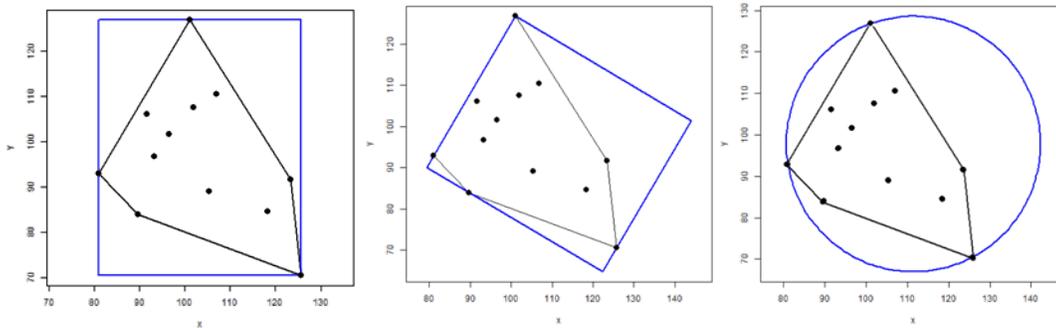


Figura 1.6: Aplicaciones de la cubierta convexa: Rectángulo envolvente más pequeño alineado con los ejes (izquierda), orientado arbitrariamente (centro), y círculo envolvente más pequeño (derecha) de un conjunto de puntos.

determinan la cubierta convexa  $CH(S)$  del conjunto dado  $S$ , calculan los vértices de  $CH(S)$  de tal manera que construyen paso por paso la polilínea que conecta los vértices de  $CH(S)$  en forma ordenada y al final es cerrada para formar la frontera del polígono  $CH(S)$ . Esta frontera del polígono  $CH(S)$  es una curva de Jordan, y la secuencia ordenada de los puntos que forma esta polilínea, corresponde a cierto sentido de trazado de la curva.

Una curva de Jordan en el plano puede ser trazada solamente en dos sentidos: en sentido horario (sentido matemático negativo) o en sentido antihorario (sentido matemático positivo), tal como se muestra en la Figura 1.7.

En esta tesis se recalca la idea de que para formular cualquier algoritmo que pretenda determinar la cubierta convexa, se deben fijar de antemano tanto un sentido de trazado de las curvas involucradas en el algoritmo como también un sistema de coordenadas del plano. Asumir cierto sentido de trazado de curva es importante para poder efectuar correctamente estudios sobre convexidad y concavidad local. Fijar un sistema de coordenadas hace posible usar correctamente herramientas como determinantes o representaciones analíticas de propiedades geométricas. Asimismo se menciona que en varias publicaciones internacionales de algoritmos, estas condiciones iniciales no son claramente dadas, lo cual causa confusión y dificulta al lector el entendimiento.

En esta tesis se trabajará siempre con el sistema de coordenadas cartesiano estándar de  $\mathbb{R}^2$ . Para todo algoritmo que se analiza o desarrolla en este trabajo, se asumirá que todas las curvas involucradas son trazadas en sentido horario. Esto se refiere en particular a la curva correspondiente a la polilínea que se construye para formar la frontera de  $CH(S)$  (datos de salida). Pero en el caso en que el conjunto  $S$  (datos de entrada) sea dado como una secuencia ordenada de vértices correspondiente a la frontera de un polígono, el sentido horario de trazado es asumido también para esta secuencia.

En general, en el proceso de la construcción de la polilínea correspondiente a la frontera de  $CH(S)$ , todos los puntos de  $S$  son analizados (uno a la vez), y luego descartados si no resultan ser vértices de  $CH(S)$ . El análisis local para decidir si un punto se agrega o no a la polilínea en construcción, se basa en revisiones locales sobre convexidad y concavidad. En la siguiente sección se explican estas propiedades geométricas.

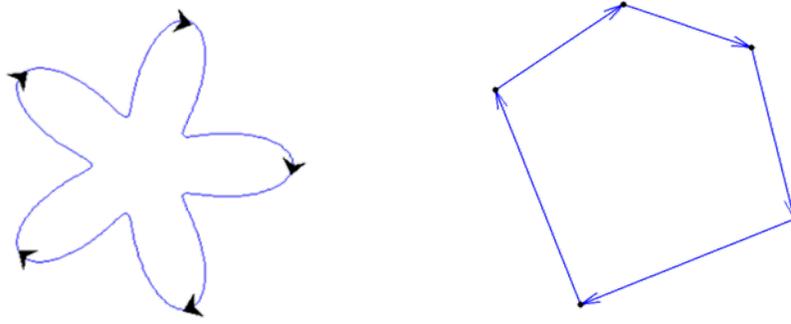


Figura 1.7: Trazado de curvas de Jordan y de polígonos en sentido horario.

### 1.5.2. Sentidos de giro y convexidad/concavidad local

Es bien conocido que para un triángulo  $T$  formado por los puntos  $p_1 = (x_1, y_1)$ ,  $p_2 = (x_2, y_2)$  y  $p_3 = (x_3, y_3)$  en el plano, su área puede ser calculada por:

$$\mathcal{A}(T) = \frac{1}{2} \cdot |D(p_1, p_2, p_3)|$$

donde  $D(p_1, p_2, p_3)$  es el determinante

$$D(p_1, p_2, p_3) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

Este determinante resulta ser positivo o negativo, dependiendo de la orientación de la tripleta ordenada de puntos  $(p_1, p_2, p_3)$ .

Los puntos  $p_1$  y  $p_2$  definen una línea recta orientada de  $p_1$  hacia  $p_2$  que divide a  $\mathbb{R}^2$  en dos semiplanos. Usando la orientación de  $p_1$  hacia  $p_2$ , podemos considerar al *semiplano derecho* y al *semiplano izquierdo* (Figura 1.8).

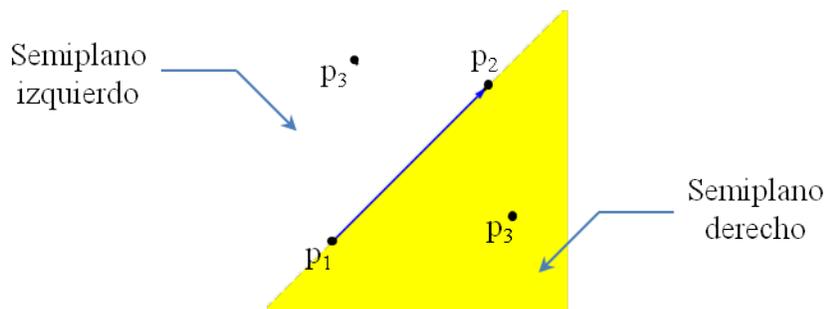


Figura 1.8: Semiplanos generados por la recta orientada desde  $p_1$  hacia  $p_2$ .

Si  $p_3$  se encuentra en el semiplano derecho, se dice que  $(p_1, p_2, p_3)$  forman una ***vuelta derecha***, y si el punto  $p_3$  se encuentra en el semiplano izquierdo, entonces la tripleta  $(p_1, p_2, p_3)$  forma una ***vuelta izquierda***.

En el sistema de coordenadas estándar de  $\mathbb{R}^2$  que hemos fijado, el determinante  $D(p_1, p_2, p_3)$  tiene signo positivo si y solo si la tripleta  $(p_1, p_2, p_3)$  forma una vuelta izquierda; y un determinante negativo es equivalente a una vuelta derecha. Si el determinante es cero significa que los tres puntos son colineales.

El tipo de vuelta también puede ser caracterizado mediante ángulos. El ángulo  $\sphericalangle(p_1, p_2, p_3)$  medido en sentido horario desde  $\overline{p_1p_2}$  hacia  $\overline{p_2p_3}$  también puede dar una referencia del tipo de vuelta. Si  $\sphericalangle(p_1, p_2, p_3)$  es menor a  $180^\circ$  es una vuelta izquierda, si es mayor es una vuelta derecha, y si es igual a  $180^\circ$  entonces los tres puntos son colineales. Cabe mencionar que el cálculo de ángulos involucra el uso de funciones trigonométricas.

La necesidad de conocer el tipo de vuelta de una tripleta de puntos surge en algoritmos que determinan la cubierta convexa de una secuencia ordenada de puntos. A partir del tipo de vuelta se puede deducir si la polilínea de entrada forma localmente una región convexa o cóncava. Bajo la suposición que todas las curvas son trazadas en sentido horario, una vuelta derecha de una tripleta  $(p_1, p_2, p_3)$  significa que los puntos  $(p_1, p_2, p_3)$  forman una convexidad local. De igual forma, una vuelta izquierda implica que se forma una concavidad local. En la Figura 1.9 se resumen algunos métodos que permiten determinar la convexidad o concavidad local.

Cabe mencionar, que no todas estas correspondencias entre vueltas y convexidad o concavidad local aplican en el caso del trazado de la polilínea en sentido antihorario.

De la Figura 1.9 resulta evidente que cuando los puntos  $(p_1, p_2, p_3)$  forman una concavidad local,  $p_2$  no puede ser un vértice de la cubierta convexa de la polilínea. Sin embargo, cuando  $p_2$  es el punto central de una convexidad local, no hay garantía alguna de que  $p_2$  sea un vértice de la cubierta convexa. Esto da una idea sobre las dificultades en la determinación de la cubierta convexa.

### 1.5.3. Descripción de algunos algoritmos representativos

A continuación se describen brevemente algunos de los algoritmos más conocidos propuestos en la literatura para determinar la cubierta convexa de un conjunto  $S$  de  $n$  puntos, donde en general no se requiere que los puntos en  $S$  estén ordenados de antemano.

Todos los algoritmos descritos en esta sección, incluyendo su pseudocódigo y una descripción más detallada pueden ser consultados en [7].

- **R. Graham (1972).**- Este algoritmo inicia con la elección de un punto arbitrario al interior del conjunto de puntos y formando un segmento de recta con un punto extremo del conjunto. Después se ordenan los demás puntos de acuerdo al ángulo formado con respecto a esta línea recta. Posteriormente se verifica la convexidad o concavidad local de tripletas de puntos consecutivos del conjunto ya ordenado. Secuencialmente, solo se conservan las tripletas de puntos que forman convexidades. La complejidad en tiempo de este algoritmo mejora a la del método anterior al ser  $\mathcal{O}(n \cdot \log n)$ .
- **J. Sklansky (1972).**- El algoritmo de Graham depende del cálculo de ángulos para ordenar y para determinar la convexidad local de una tripleta de puntos. Estas operaciones fueron aceleradas mediante el uso del determinante tal como se detalla en la Sección 1.5.2. El uso del determinante para tal efecto es conocido como la *prueba de Sklansky* y evita

CONVEXIDAD LOCAL	CONCAVIDAD LOCAL	COLINEALIDAD
Por ángulo		
<p><math>\alpha &gt; 180^\circ</math></p>	<p><math>\alpha &lt; 180^\circ</math></p>	<p><math>\alpha = 180^\circ</math></p>
Por determinante		
$D(p_1, p_2, p_3) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_2 & y_3 & 1 \end{vmatrix} < 0$	$D(p_1, p_2, p_3) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_2 & y_3 & 1 \end{vmatrix} > 0$	$D(p_1, p_2, p_3) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_2 & y_3 & 1 \end{vmatrix} = 0$
Por tipo de vuelta		
<p><i>Vuelta derecha</i></p>	<p><i>Vuelta izquierda</i></p>	<p><i>No hay vuelta</i></p>
* El trazado de la polilínea es en sentido horario		

Figura 1.9: Tabla comparativa de métodos para determinar la concavidad o convexidad local de una tripleta de puntos.

el cálculo de funciones trigonométricas. Aunque este método acelera la ejecución, tiene la misma complejidad en tiempo que el algoritmo de Graham.

- **Quickhull (1977).**- Este algoritmo está basado en el principio *divide y vencerás*, el cual fue utilizado primeramente para desarrollar el ampliamente conocido algoritmo de ordenamiento *Quicksort*. *Quickhull* consiste en dividir el conjunto original de puntos mediante una línea que une los puntos horizontalmente extremos. Después se forma un triángulo uniendo el segmento de recta obtenido previamente con el punto más lejano a ella. Los puntos en el interior de este triángulo se descartan, y se forman nuevos triángulos utilizando como base los lados añadidos en el paso anterior. Este procedimiento se realiza de forma recursiva. La complejidad esperada de este algoritmo es de  $\mathcal{O}(n \cdot \log n)$ , y de  $\mathcal{O}(n^2)$  en el peor caso.
- **Incremental (1984).**- El algoritmo incremental consiste en tomar dos puntos conocidos de la cubierta convexa y formar con ellos un segmento de recta que funciona como polilínea inicial, es decir, un lado inicial. Se elige un sentido de trazado de la polilínea a construir para formar la frontera de la cubierta convexa, e.g. sentido horario. Se forma un nuevo lado de la polilínea con uno de los dos puntos iniciales y cada uno de los puntos restantes del conjunto. Cuando el nuevo segmento de recta divide al plano de tal forma que todos los puntos restantes quedan en uno de los semiplanos (en este caso, el derecho), se agrega este segmento como lado a la polilínea en construcción, y el proceso se repite con el último punto agregado y los sobrantes. La complejidad en tiempo de este algoritmo es  $\mathcal{O}(n \cdot \log n)$ .
- **Rubberband.**- Este método consiste en ordenar los puntos del conjunto dado en forma ascendente de acuerdo a su coordenada horizontal, y de forma descendente con respecto a su coordenada vertical si hubiera varios puntos con la misma abscisa. En un segundo recorrido, se verifica la convexidad o concavidad local de tripletas de puntos comenzando por el extremo izquierdo de la polilínea, tomando en cuenta los ángulos formados en la parte inferior de la polilínea. De esta manera se formará la parte inferior de la cubierta convexa, denominada *arco convexo inferior*. En un tercer recorrido se hace un procedimiento similar al anterior en la parte superior de la polilínea, partiendo del extremo derecho. Con ello se obtendrá el llamado *arco convexo superior*, y con este se completará la cubierta convexa. Este algoritmo tiene una complejidad de  $\mathcal{O}(n \cdot \log n)$  en el peor caso.

Como se puede apreciar, diversos métodos que obtienen la cubierta convexa a partir de un conjunto de puntos requieren de un algoritmo de ordenamiento (*sorting*). Los mejores algoritmos de ordenamiento, cuando son aplicados a un conjunto de  $n$  elementos, tienen una complejidad de  $\mathcal{O}(n \cdot \log n)$ , lo cual acota el desempeño de los algoritmos que utilizan como datos de entrada un conjunto de puntos no ordenados para determinar su cubierta convexa.

Se recuerda que la cubierta convexa de un polígono en el plano es igual a la de su conjunto de vértices. En muchas aplicaciones, los vértices de un polígono son dados como una secuencia ordenada de puntos, donde el orden corresponde a la secuencia de trazado de la frontera del polígono. Este orden puede ser aprovechado por algoritmos que determinan la cubierta convexa del polígono, los cuales ya no necesitarían ordenar los puntos. En consecuencia, la cubierta

convexa puede ser determinada en tiempo lineal, es decir, mediante algoritmos que tienen una complejidad en tiempo de orden  $\mathcal{O}(n)$ .

Una característica importante de los polígonos que puede influir en la determinación de su cubierta convexa es la *visibilidad desde el exterior* (Figura 1.10), definida como sigue:

**Definición 1.5.1** *Un polígono simple  $P$  es visible desde el exterior si para cualquier punto  $q$  en la frontera de  $P$  existe un rayo con  $q$  como punto inicial y apuntando hacia el exterior de  $P$ , que no interseca al polígono en algún otro punto además de  $q$ .*



Figura 1.10: Visibilidad desde el exterior: polígono visible (izquierda) y no visible (derecha). Cualquier rayo que parta de  $q$  hacia el exterior interseca a  $P$ .

Algunos algoritmos que determinan la cubierta convexa de polígonos en el plano, y que aprovechan que el polígono está dado por una secuencia ordenada de sus vértices, se describen en la siguiente lista.

- **J. Sklansky (1972).**- Al tener los puntos de entrada dados como secuencia ordenada, el algoritmo solo ejecuta la última parte del algoritmo de Graham, incluyendo la mejora de utilizar el determinante para evitar el uso de ángulos y funciones trigonométricas. Existen propuestas de implementación eficientes de este algoritmo que incluyen el uso de estructuras tipo pila para agregar y descartar vértices. Sin embargo, este algoritmo funciona únicamente para polígonos visibles desde el exterior.
- **R. Klette (1983).**- Este algoritmo resuelve la restricción del algoritmo de Sklansky, ya que permite detectar cuando una secuencia de vértices se encuentra en una cavidad no convexa. Por ello, el algoritmo funciona correctamente para cualquier polígono simple como dato de entrada.
- **A. Melkman (1987).**- La característica principal de este algoritmo es el uso de la estructura de datos llamada *deque*, cuyas propiedades se describieron en la Subsección 1.3.2. El pseudocódigo y una explicación más a fondo de la operación de este algoritmo se presenta en la Sección 1.5.4. Asimismo, la fuente original se presenta en el apéndice C. La importancia de este algoritmo radica en que es un componente fundamental de la propuesta del algoritmo para determinar la cubierta convexa relativa, estudiado en esta tesis.

---

**Entrada:** Una polilínea simple  $\rho = (p_0, p_1, \dots, p_{n-1})$  con  $n$  vértices en el plano.

**Salida:** La lista ordenada de los vértices de la cubierta convexa de la polilínea de entrada.

- 1: PUSH  $p_0$  y luego PUSH  $p_1$  en la *deque*. PUSH  $p_2$  e INSERT  $p_2$ . Los tres vértices deben formar un triángulo. Sea  $k = 3$ .
- 2: **while**  $k < n$  **do**
- 3:     **while**  $D(q_{bot}, q_{bot+1}, p_k) > 0$  AND  $D(q_{top-1}, q_{top}, p_k) > 0$  AND  $k < n$  **do**
- 4:          $k = k + 1$
- 5:     **end while**
- 6:     **while**  $D(q_{bot}, q_{bot+1}, p_k) \leq 0$  **do**
- 7:         DELETE
- 8:     **end while**
- 9:     INSERT  $p_k$
- 10:    **while**  $D(q_{top-1}, q_{top}, p_k) \leq 0$  **do**
- 11:        POP
- 12:    **end while**
- 13:    PUSH  $p_k$ ,  $k = k + 1$
- 14: **end while**
- 15: Los vértices de la cubierta convexa se encuentran en la *deque*.

---

Figura 1.11: Pseudocódigo del algoritmo de Melkman [7].

#### 1.5.4. Algoritmo de Melkman

El algoritmo de Melkman es muy eficiente para determinar la cubierta convexa de una polilínea. En algunas fuentes, este algoritmo es reportado como el mejor para determinar la cubierta convexa [15]. Su dato de entrada es una polilínea simple (abierta) dada como la secuencia ordenada de sus  $n$  vértices  $\rho = (p_0, p_1, \dots, p_{n-1})$ . En esta tesis, esta polilínea se considera trazada en sentido horario. Su dato de salida es una *deque*, que incluye los  $m$  vértices de la cubierta convexa de la polilínea de entrada en forma de una polilínea cerrada, definida como  $d = (q_m, q_1, \dots, q_{m-1}, q_m)$ .

La característica más significativa de este algoritmo es el uso de la *deque*, la cual permite agregar y eliminar elementos por sus dos extremos, a diferencia de otras estructuras como las pilas y colas. Esta particularidad permite determinar *en línea* la cubierta convexa de una polilínea, ya que al realizar el análisis de cada punto, el resultado actual es siempre la cubierta convexa (en forma de polilínea cerrada) de todos los puntos analizados hasta ese momento.

Este algoritmo calcula la cubierta convexa de una polilínea simple en tiempo lineal. Cada vértice es procesado una sola vez, de igual forma que en el algoritmo de J. Sklansky o de R. Klette, sin embargo, una de las diferencias más importantes con respecto a ellos es que no es necesario conocer de antemano algún punto que pertenezca a la cubierta convexa. A continuación se explica brevemente el funcionamiento del algoritmo con ayuda del pseudocódigo mostrado en la Figura 1.11.

Primeramente, el algoritmo forma una *deque* inicial con los primeros tres vértices de la

polilínea de entrada, ingresando un punto a la vez por el extremo superior (*top*) y añadiendo el tercer punto también por el extremo inferior (*bottom*). Esto genera una polilínea cerrada que representa un polígono convexo inicial que se irá modificando al agregarse los vértices correspondientes a la cubierta convexa de la polilínea de entrada. Este polígono inicial consta de tres vértices debido a que es la cantidad mínima de vértices que integran a un polígono (convexo).

Posteriormente, el algoritmo analiza cada vértice restante de la polilínea inicial mediante un proceso iterativo principal, verificando un vértice a la vez en cada ciclo.

Dentro de este proceso principal (línea 2 a 14 en Figura 1.11), existen tres procesos iterativos secundarios, los cuales identifican la posición del vértice a verificar ( $p_k$ ) con respecto a la cubierta convexa determinada hasta ese momento. La forma en que se realiza esta operación se describe a continuación.

El primero de estos tres procesos (línea 3 a 5 en Figura 1.11) verifica si el vértice  $p_k$  se encuentra dentro de la cubierta convexa ya determinada hasta el momento. Si es así, el vértice se descarta y se pasa al siguiente vértice de la polilínea de entrada ( $p_{k+1}$ ). Esta verificación se hace validando dos condiciones de forma simultánea. La primera condición es que el vértice  $p_k$  se encuentre a la izquierda de la línea  $\overline{q_{bot}, q_{bot+1}}$ , y la segunda es que se encuentre a la izquierda de la línea  $\overline{q_{top}, q_{top-1}}$ , donde  $q_{top}$  y  $q_{bot}$  es el elemento extremo superior y el elemento extremo inferior de la *deque*, respectivamente.

Después de este proceso, el algoritmo continua con otro proceso iterativo (línea 6 a 8 en Figura 1.11), el cual se encarga de revisar si el vértice  $p_k$  se encuentra a la derecha de la línea  $\overline{q_{bot}, q_{bot+1}}$ . Si es así, se realiza una operación *DELETE*, eliminando el elemento extremo inferior de la *deque*. Después se verifica si el vértice  $p_k$  está a la derecha de  $\overline{q_{bot+1}, q_{bot+2}}$  y así sucesivamente hasta que  $p_k$  se encuentra a la izquierda.

Si la condición del ciclo se cumple, también implica que el vértice  $p_k$  no se encuentra dentro de la cubierta convexa determinada hasta ese momento, por tanto el vértice  $p_k$  tiene que ser ingresado a la *deque*. Esta operación se realiza por el extremo inferior (*INSERT*) ya que en este proceso se analizan los vértices de la sección inferior (*bottom*) de la polilínea resultante.

El tercer proceso iterativo secundario (línea 10 a 12 en Figura 1.11) realiza una revisión análoga a la del proceso anterior. Esta vez se verifica si el vértice  $p_k$  está a la derecha de la línea  $\overline{q_{top-1}, q_{top}}$ . Puesto que se examinan vértices del extremo superior de la polilínea resultante, las operaciones que se realizan son *POP* y *PUSH*, respectivamente con respecto al proceso previo.

Una vez que se agotan los vértices de la polilínea de entrada para procesar, el algoritmo finaliza y en la *deque* se tendrán los vértices de la cubierta convexa en forma de una polilínea cerrada.

# Capítulo 2

## Cubierta convexa relativa

### 2.1. Definición y propiedades de la cubierta convexa relativa

La cubierta convexa relativa generaliza el concepto de la cubierta convexa [7]. Tal como se estableció en el capítulo anterior, a lo largo del presente capítulo se asume que  $A$  y  $B$  son dos polígonos simples que cumplen que  $A \subseteq B \subset \mathbb{R}^2$ . En esta situación, al polígono  $A$  se le denominará **polígono interior**, y a  $B$  **polígono exterior**. El polígono  $A$  está dado por sus  $n$  vértices, tal que  $A = \langle p_1, p_2, \dots, p_n \rangle$ . Similarmente,  $B$  está dado por sus  $m$  vértices y por lo tanto se denotará por  $B = \langle q_1, q_2, \dots, q_m \rangle$ .

**Definición 2.1.1** Sean  $M, N \subset \mathbb{R}^2$  dos subconjuntos cualesquiera no vacíos que cumplen que  $M \subseteq N$ . Entonces  $M$  se llama  **$N$ -convexo** si cualquier segmento de línea recta en  $N$  que tiene ambos puntos extremos en  $M$ , también está contenido en  $M$ .

**Definición 2.1.2** Sean  $M, N \subset \mathbb{R}^2$  dos subconjuntos cualesquiera no vacíos que cumplen que  $M \subseteq N$ . La **cubierta convexa de  $M$  relativa a  $N$** , denotada por  $CH_N(M)$ , se define como la intersección de todos los subconjuntos de  $\mathbb{R}^2$  que contienen a  $M$  y son  $N$ -convexos.  $CH_N(M)$  también es conocida como la **cubierta convexa relativa** de  $M$  con respecto a  $N$ .

Si el conjunto  $M$  es convexo entonces es también  $N$ -convexo, tomando en cuenta que  $M \subseteq N$ . Asimismo, es evidente que cualquier conjunto  $M$  es también  $M$ -convexo.

En este trabajo es de interés aplicar esta definición únicamente a los polígonos  $A$  y  $B$  definidos inicialmente, para estudiar la cubierta convexa relativa  $CH_B(A)$ . Dado que la intersección de dos polígonos es también un polígono, la cubierta convexa relativa de  $A$  con respecto a  $B$  es un polígono (Figura 2.1).

Otra observación importante sobre la cubierta convexa relativa  $CH_B(A)$ , que es análoga a una propiedad de la cubierta convexa, es la siguiente:

**Lema 2.1.3** Para cualquier polígono  $A$ , si  $V$  denota al conjunto de los vértices de  $M$ , entonces  $CH_B(V) = CH_B(A)$  [7].

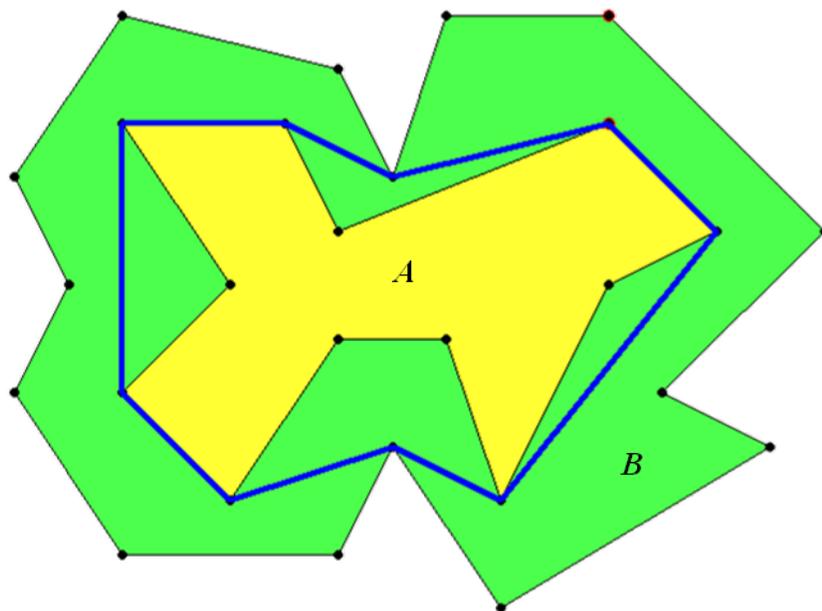


Figura 2.1: Cubierta convexa relativa  $CH_B(A)$ .

Esto da lugar a soluciones algorítmicas para la cubierta convexa relativa  $CH_B(A)$ , para polígonos  $A, B$  tales que  $A \subseteq B$ .

Para el desarrollo de algoritmos que determinan la cubierta convexa relativa  $CH_B(A)$ , con los polígonos  $A$  y  $B$  como datos de entrada, se aprovechan algunas propiedades importantes de  $CH_B(A)$ , las cuales se resumen a continuación.

Sabiendo que  $A$  y  $B$  son polígonos simples en  $\mathbb{R}^2$  tales que  $A \subseteq B$ , y que  $CH(A)$  denota la cubierta convexa de  $A$ , se tiene lo siguiente:

1. Solo vértices convexos del polígono  $A$  y solo vértices cóncavos del polígono  $B$  son candidatos para ser vértices de la cubierta convexa relativa  $CH_B(A)$  [12] [13, p. 124].
2. Si  $CH(A) \subseteq B$  o si  $B$  es convexo, entonces  $CH_B(A) = CH(A)$ .
3. Todos los vértices de la cubierta convexa  $CH(A)$  son vértices de  $CH_B(A)$  [2, p. 266].

## 2.2. La cubierta convexa relativa y el polígono de longitud mínima

En esta sección se describirá la estrecha relación que guarda la cubierta convexa relativa con el llamado polígono de perímetro mínimo o polígono de longitud mínima. A partir de ella se expondrá una de las aplicaciones más importantes de la cubierta convexa relativa dentro del plano digital, que es la estimación de la longitud de una curva a partir su versión digitalizada.

Hay que recordar que la frontera de todo polígono simple es una curva de Jordan, y que el interior de una curva de Jordan es la región acotada por la curva. Entonces, el *polígono de*

**perímetro mínimo de  $A$  (con respecto a  $B$ )** (**MPP**, por sus siglas en inglés) se define como la cerradura del interior de la curva de Jordan de menor longitud, entre todas las curvas de Jordan contenidas en el conjunto anular  $(B \setminus A^\circ)$  que circunscribe al polígono  $A$ . Se ha demostrado que el **MPP** de  $A$  siempre existe, y está determinado de forma única [12] [13, p. 125].

Recordando también que un homeomorfismo es una biyección continua cuyo mapeo inverso es continuo también, es posible definir ahora a una **región anular** como todo conjunto homeomorfo a la región abierta entre dos circunferencias concéntricas [10], por ejemplo:

$$\{(x, y) \in \mathbb{R}^2 : r < (x^2 + y^2)^{1/2} < R\}.$$

En el contexto del análisis de imágenes digitales, y bajo suposiciones especiales referentes a los polígonos  $A$  y  $B$ , el **MPP** es conocido como **polígono de longitud mínima** (**MLP**, por sus siglas en inglés) [4] [5] [6]. Dichas condiciones están relacionadas con la digitalización de curvas (de Jordan), donde el **MLP** es utilizado para estimar la longitud de una curva a partir de su curva digital correspondiente [12] [13], tal como se explica a continuación.

Para desarrollar un concepto de curva digital correspondiente a una curva de Jordan dada  $\gamma$ , se adopta el modelo más conocido del plano digital dado como una retícula regular  $\mathbb{G}$ , también conocida como rejilla o cuadrícula. En esta tesis se considera el caso estándar donde  $\mathbb{G}$  consiste en cuadrados del mismo tamaño cuyos interiores son disjuntos (por pares) y cuya unión cubre al plano  $\mathbb{R}^2$ . Todos estos cuadrados tienen la misma longitud de lado  $l$ , la cual se denomina **constante de retícula**, mientras que a su recíproco  $h = 1/l$  se le llama **resolución (de digitalización)** [5]. En esta tesis se asume que  $h$  es un número natural, tal que  $l$  es una fracción de 1. Entonces, para una resolución  $h$  dada, cada uno de estos cuadrados puede definirse como:

$$\mathbb{G}_{(w_1, w_2)}^h = \{x \in \mathbb{R}^2 : w_i \leq x_i \leq w_i + l, i = 1, 2\}$$

con  $w_1 \in \{a, a + l, a + 2l, \dots, a + (h - 1)l\}$  para  $a$  entero arbitrario, y similarmente  $w_2 \in \{b, b + l, b + 2l, \dots, b + (h - 1)l\}$  para  $b$  entero arbitrario.

En la figura 2.2 se muestran rejillas ortogonales con diferentes resoluciones: la primera con resolución  $h = 1$ , la segunda con  $h = 2$ , y la tercera  $h = 4$ .

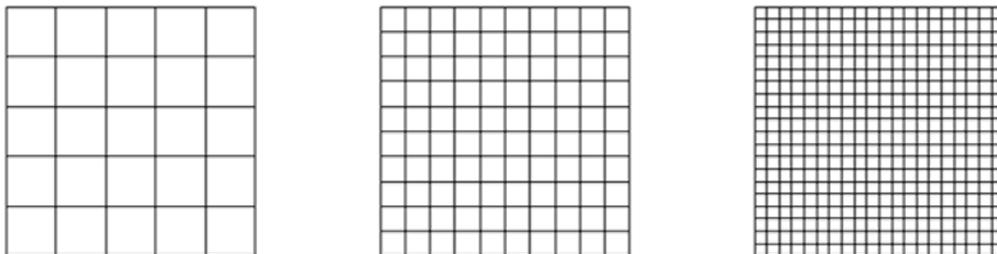


Figura 2.2: Retículas con diferente resolución.

Existen diversos métodos de digitalización de subconjuntos del plano euclidiano. En el con-

texto de curvas digitales y de la estimación de longitud de curva que se describe aquí, la digitalización de Jordan es de interés, y se define a continuación.

**Definición 2.2.1** Sea  $S$  un subconjunto no vacío de  $\mathbb{R}^2$ , y  $\mathbb{G}^h$  la retícula ortogonal con una resolución  $h$ . Al conjunto  $J_h^-(S)$  dado por la unión de todos los cuadrados que están completamente contenidos en  $S$ , se le denomina **digitalización interior de Jordan** de  $S$ , y al conjunto  $J_h^+(S)$  dado por la unión de todos los cuadrados que tienen intersección no vacía con  $S$ , se le llama **digitalización exterior de Jordan** de  $S$  [5, p. 59].

En la Figura 2.3 se muestra un ejemplo de la digitalización de Jordan de un conjunto  $S$ .

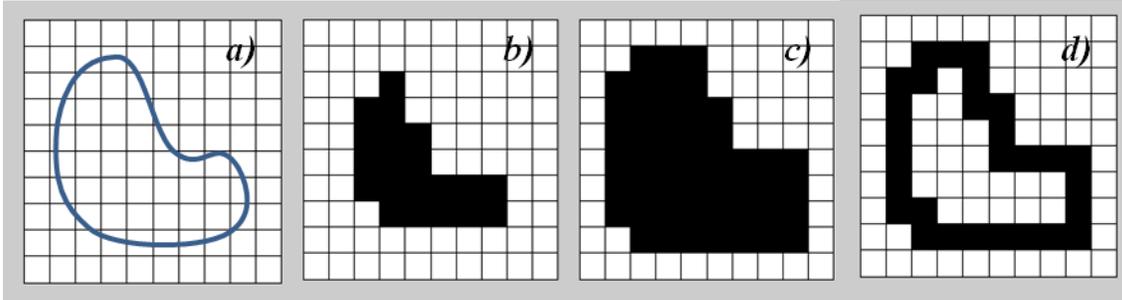


Figura 2.3: a) Conjunto  $S$  y su frontera  $\gamma$ , b) digitalización interior  $J_h^-(S)$ , c) digitalización exterior  $J_h^+(S)$  y d) curva digital  $\Gamma$ .

Cuando un conjunto  $S$  se digitaliza, el resultado de este proceso tiene propiedades muy particulares. Una de ellas, que se mencionará a lo largo de esta sección, se define como sigue:

**Definición 2.2.2** Dentro de un sistema de coordenadas cartesianas, se dice que una línea es **isotética** si es paralela a un eje coordenado. A su vez, un **polígono isotético** es aquél que está formado solo por lados isotéticos.

Dado que la digitalización (interior o exterior de Jordan) de cualquier subconjunto  $S$  del plano es una unión de cuadrados, esta digitalización de  $S$ , vista como subconjunto del plano euclidiano es un polígono isotético. Suponiendo una curva de Jordan  $\gamma : [0, 1] \rightarrow \mathbb{R}^2$ , y denotando su interior como  $S$ , entonces la **curva digital** correspondiente a  $\gamma$  se construye mediante la digitalización (interior y exterior de Jordan) de  $S$  (Figura 2.3).

En esta tesis, bajo curva digital se entiende siempre una curva de Jordan digital.

**Definición 2.2.3** Para cualquier curva de Jordan  $\gamma : [0, 1] \rightarrow \mathbb{R}^2$ , denotando su interior como  $S$ , y suponiendo que la resolución de la retícula ortogonal es suficientemente grande tal que  $S$  contiene al menos un cuadrado, la **curva de Jordan digital**  $\Gamma$  correspondiente a  $\gamma$  se define como la diferencia entre la digitalización exterior de Jordan de  $S$  y la digitalización interior de Jordan de  $S$ :

$$\Gamma = J_h^+(S) \setminus J_h^-(S) .$$

En consecuencia, la curva digital  $\Gamma$  correspondiente a una curva de Jordan  $\gamma$  es un conjunto especial de cuadrados. En la Figura 2.3 se puede ver que  $\Gamma$  tiene la propiedad especial de que todos sus cuadrados pueden ser ordenados en una lista cíclica y que, en esta lista, cada cuadrado tiene un lado común con su sucesor y un lado común con su antecesor, pero no se interseca con otro cuadrado de la lista. Un conjunto de cuadrados con esta propiedad especial también es conocido como *continuo digital* [13], y esta propiedad es exigida en la literatura a cada curva digital.

La curva digital  $\Gamma$ , al ser un conjunto de cuadrados, también puede ser interpretado como un subconjunto  $C$  del plano euclidiano, al unificar todos sus cuadrados. Así, este conjunto  $C$  tiene las siguientes características:

- $C$  es un conjunto anular.
- La frontera interior de  $C$  es una curva de Jordan, y es la frontera de un polígono isotético  $A$ , correspondiente a  $J_h^-$ .
- La frontera exterior de  $C$  es una curva de Jordan, y es la frontera de un polígono isotético  $B$ , correspondiente a  $J_h^+$ .
- La cubierta convexa relativa  $CH_B(A)$  coincide con el *MLP* de  $A$  relativo a  $B$  [5] (Figura 2.4).
- La cubierta convexa relativa  $CH_B(A)$  está bien definida y es un polígono. La frontera del conjunto  $CH_B(A)$  es una curva de Jordan que se encuentra dentro del conjunto anular  $C$ . La longitud de dicha curva es una aproximación de la longitud de la curva  $\gamma$  (Figura 2.5). En [13] fue demostrado que en un proceso iterativo de digitalizaciones con resoluciones cada vez mayores, la secuencia de las longitudes de las fronteras de las cubiertas convexas relativas  $CH_B(A)$ , converge a la longitud verdadera de la curva  $\gamma$ , bajo ciertas condiciones. Este concepto de convergencia posteriormente se definirá de manera formal.
- Las fronteras  $fr(C_1)$  y  $fr(C_2)$  se encuentran a una distancia Hausdorff igual a uno (una de la otra) con respecto a la distancia  $d_\infty$ . Recordando que

$$d_\infty((x_1, x_2), (y_1, y_2)) = \text{máx}\{|x_1 - y_1|, |x_2 - y_2|\}, (x_1, x_2), (y_1, y_2) \in \mathbb{R}^2,$$

se tiene entonces que

$$\text{máx}\{\sup\{\inf\{d_\infty(x, y) : y \in fr(C_2)\} : x \in fr(C_1)\}, \sup\{\inf\{d_\infty(x, y) : x \in fr(C_1)\} : y \in fr(C_2)\}\} = 1.$$

Históricamente, se han desarrollado diversos métodos para estimar la longitud de una curva  $\gamma$  a partir de su versión digitalizada. De forma general, un estimador busca una curva poligonal que tenga un perímetro que se aproxime al perímetro real de la curva digitalizada.

Actualmente, el enorme progreso tecnológico en el ámbito de imágenes digitales ha facilitado la obtención de imágenes de gran resolución. En este sentido, es de utilidad el uso de estimadores que tienen la propiedad de **convergencia bajo aumento de resolución**, la cual se define a continuación [4] [5]:

**Definición 2.2.4** Sea  $\mathbb{F}$  una familia de subconjuntos  $S$  de  $\mathbb{R}^2$ , y sea  $dig_h$  un método de digitalización aplicable a todos los  $S \in \mathbb{F}$ , con una resolución  $h$ . Además, sea  $Q$  una propiedad numérica definida para todos  $S \in \mathbb{F}$  (por ejemplo longitud o área). Un estimador  $E_Q$  de la propiedad  $Q$  se llama **convergente bajo aumento de resolución** para  $\mathbb{F}$  y para  $dig_h$  si para cualquier  $S \in \mathbb{F}$ , existe una resolución  $h_S > 0$  tal que el valor estimado  $E_Q(dig_h(S))$  queda definido para cualquier resolución  $h \geq h_S$ , y

$$|E_Q(dig_h(S)) - Q(S)| \leq k(h)$$

donde  $k(h)$  es una función definida en los números reales que converge a cero cuando  $h \rightarrow \infty$ .

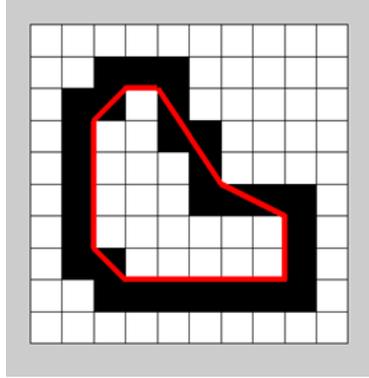


Figura 2.4: La cubierta convexa relativa coincide con el *MLP*.

Para un estimador de longitud de curva basada en el *MLP*, una curva de Jordan  $\gamma$  con interior  $S$  es representada por la curva digital  $\Gamma = J_h^+(S) \setminus J_h^-(S)$ .

## 2.3. Aplicaciones de la cubierta convexa relativa

Tal como se mencionó en la sección anterior, una de las principales aplicaciones de la cubierta convexa relativa de interés en el análisis de imágenes digitales es la estimación de la longitud de una curva, a partir de su versión digitalizada, ya que ofrece un método convergente bajo aumento de resolución. En la Figura 2.5 se muestra un ejemplo simple de un conjunto digitalizado y de la cubierta convexa relativa utilizada para aproximar la longitud de la frontera de tal conjunto.

Otra posible aplicación de la cubierta convexa relativa es la planeación del camino más corto para un robot móvil dentro de un ambiente bidimensional con restricciones. En este sentido, el polígono exterior puede quedar definido por la frontera del área disponible en la cual se puede mover un robot, y el polígono interior se forma con las esquinas de un obstáculo al interior del área disponible. En la Figura 2.6 se muestra una concepción básica de esta aplicación, presentada en el *First Robot Marathon*, celebrado en Osaka, Japón en 2011 [18] [19].



## 2.4. Algoritmos que determinan la cubierta convexa relativa

El desarrollo de algoritmos para determinar la cubierta convexa relativa ha sido un tema de investigación históricamente más reciente que el cálculo de la cubierta convexa. A continuación se presenta una breve descripción de algunos algoritmos propuestos en la literatura. Es importante mencionar que de los siguientes algoritmos, el propuesto por G.T. Toussaint, y por G. Klette funcionan para el caso general, es decir, para dos polígonos simples arbitrarios  $A$  y  $B$  en el plano tal que  $A \subseteq B$ . En contraste, el algoritmo propuesto por R. Klette y V. Kovalevsky, solo aplica para curvas digitales, las cuales fueron descritas en la Sección 2.2.

- **Toussaint (1986).**- El autor relaciona el problema de la cubierta convexa relativa con el del camino más corto entre dos vértices de un polígono. Desde esta perspectiva, se considera un vértice extremo (es decir, con coordenada máxima o mínima en sentido vertical u horizontal) del polígono interior como vértice inicial y final de un nuevo polígono generado a partir de  $A$  y  $B$ . Este nuevo polígono se obtiene realizando un corte en la región anular  $B \setminus A^\circ$ , introduciendo un nuevo lado (con orientación doble) que inicia en un punto extremo de  $A$  y finaliza en un vértice cercano de  $B$ . De esta manera, el polígono interior y exterior forman un único polígono. Después, el autor propone realizar un proceso de triangulación al interior de este nuevo polígono resultante  $B \setminus A^\circ$ , y seguidamente encontrar el camino más corto a través de  $B \setminus A^\circ$  a partir del punto extremo donde se realizó el corte. La triangulación puede realizarse con una complejidad en tiempo de  $\mathcal{O}(n \cdot \log(\log n))$  (donde  $n$  es la suma del número de vértices del polígono interior y exterior), y encontrar el camino más corto puede realizarse en tiempo lineal, usando esta triangulación. Sin embargo, la triangulación propuesta es un proceso complejo, y el autor solo provee un bosquejo sin detalles [7] [14].
- **Klette-Kovalevsky-Yip (1999).**- Este algoritmo, en contraste a los algoritmos de Toussaint y de G. Klette, fue desarrollado para determinar el *MLP* o equivalentemente, la cubierta convexa relativa, de una curva digital, es decir, de un continuo digital. Recordando de la Sección 2.2 las propiedades de una curva digital, el algoritmo de Klette-Kovalevsky-Yip es aplicable sólomente en esta situación. Denotando como  $A$  la frontera interior del conjunto anular  $C$ , y como  $B$  su frontera exterior, entonces  $A$  y  $B$  describen polígonos isotéticos tales que  $A \subset B$ , y la cubierta convexa relativa  $CH_B(A)$  está completamente determinada por un subconjunto de los vértices de  $A$  y  $B$ . Este algoritmo determina a los vértices de  $CH_B(A)$ , en tiempo lineal con respecto al número total de vértices de  $A$  y  $B$ . Es bien sabido ([5]) que sólomente vértices convexos de  $A$  y vértices cóncavos de  $B$  son candidatos para ser vértices de  $CH_B(A)$ . Además, en la situación especial de que  $C = B \setminus A^\circ$  forma una curva digital, existe un mapeo biyectivo entre los vértices de  $A$  y  $B$  [3], en el cual a cada vértice convexo de  $A$  corresponde un único vértice convexo de  $B$ , y de igual forma para los vértices cóncavos. Aprovechando estas propiedades, en un primer recorrido de la polilínea  $A$  en sentido anti-horario, el algoritmo registra en una lista todos los vértices de  $A$  y los etiqueta como convexos o cóncavos. Los vértices colineales no son incluidos en la lista. En este mismo recorrido de la lista, cada vértice cóncavo de  $A$  es reemplazado por su correspondiente vértice cóncavo de  $B$ . Como resultado se obtiene una

lista ordenada (correspondiente al recorrido de la polilínea  $A$ ) de todos los candidatos a ser vértices de  $CH_B(A)$ . En un segundo recorrido de la lista, el algoritmo comienza en un vértice conocido de  $CH_B(A)$  (vértice extremo), y estando en cada vértice encontrado  $p$  de  $CH_B(A)$ , determina el siguiente vértice  $q$  de  $CH_B(A)$ , como resultado de una búsqueda local. Esta búsqueda verifica si los vértices siguientes a  $p$  en la lista, tienen la propiedad que el segmento de línea recta  $\overline{pq}$  se encuentra dentro del conjunto anular  $C$ . Cuando un punto de la lista  $q$  cumple esta propiedad pero el siguiente punto de la lista ya no la cumple, entonces  $q$  es el siguiente vértice de  $CH_B(A)$ . Intuitivamente, estando en  $p$ , el vértice  $q$  es el punto más lejano en la lista tal que  $\overline{pq}$  no se sale de la región  $C$ .

- **G. Klette (2011).**- La autora propone primeramente calcular la cubierta convexa de los polígonos interior y exterior mediante el algoritmo de Melkman. Después se verifica si alguna cavidad del polígono interior tiene una intersección no vacía con alguna cavidad del polígono exterior. Si esto ocurre, se generan dos nuevos polígonos interior y exterior, denotados como  $I$  y  $O$  respectivamente. El nuevo polígono exterior  $O$  se genera con los vértices que forman la cavidad en  $A$ . El nuevo polígono interior  $I$  se genera con los puntos de la cavidad de  $B$  que se encuentran dentro de la cavidad de  $A$ , además de los vértices de la cubierta de la cavidad de  $A$ . De acuerdo a las propiedades descritas en la Sección 2.1 se tiene que los vértices de la cubierta convexa del nuevo polígono interior  $I$  son parte de la cubierta convexa relativa  $CH_B(A)$ . Esto significa que se tiene el mismo problema que al inicio, pero con polígonos más pequeños. Así, este problema se puede resolver aplicando un método recursivo, el cual simplifica el problema inicial cada que se realizan las operaciones descritas previamente hasta llegar al caso más simple, lo resuelve, y después une los resultados parciales obtenidos previamente. En esta situación, el caso más simple (la base) es cuando el polígono interior es un triángulo. Finalmente, la cubierta convexa relativa se forma uniendo los vértices de la cubierta convexa de todos los polígonos interiores de cada paso recursivo.

Una descripción más detallada de este algoritmo se encuentra en la Subsección 2.5.2 y su pseudocódigo se muestra en la Figura 2.9. La fuente original [2] se adjunta en el Apéndice B.

## 2.5. Algoritmo de G. Klette

### 2.5.1. Propiedades importantes

Las siguientes propiedades de la cubierta convexa relativa  $CH_B(A)$  sirven como base y justificación del algoritmo propuesto en [2].

**Teorema 2.5.1** *Todos los vértices de la cubierta convexa del polígono interior  $A$  son también vértices de la cubierta convexa relativa  $CH_B(A)$  [2, p. 266].*

De acuerdo al teorema anterior, es posible determinar, al menos, un subconjunto de todos los vértices de la cubierta convexa relativa  $CH_B(A)$  mediante el cálculo de los vértices de la cubierta convexa del polígono interior  $A$ .

**Teorema 2.5.2** *La cubierta convexa relativa  $CH_B(A)$  es igual a la cubierta convexa de  $A$  si la cubierta convexa de  $A$  está completamente contenida en  $B$  (i.e.  $CH(A) \subseteq B$ ) o si  $B$  es convexo [2, p. 265].*

En consecuencia, la cubierta convexa relativa  $CH_B(A)$  es diferente de  $CH(A)$  si existe al menos una cavidad  $CAV_i(A)$  de  $A$  y una cavidad  $CAV_j(B)$  de  $B$  tal que  $(CAV_i(A))^\circ \cap CAV_j(B) \neq \emptyset$ , donde  $M^\circ$  denota el interior topológico del conjunto  $M \subseteq \mathbb{R}^2$  (Figura 2.7).

Cuando se tiene este caso, es posible definir dos nuevos polígonos. Sea  $I$  el polígono definido por los vértices de la cubierta de la cavidad  $CAV_i(A)$  llamados  $p_s$  y  $p_e$ , y todos los vértices de la cavidad  $CAV_j(B)$  localizados dentro de  $CAV_i(A)$ . También se define  $O$  como el polígono formado por todos los vértices de  $A$  que forman la cavidad  $CAV_i(A)$  desde  $p_s$  hasta  $p_e$ .

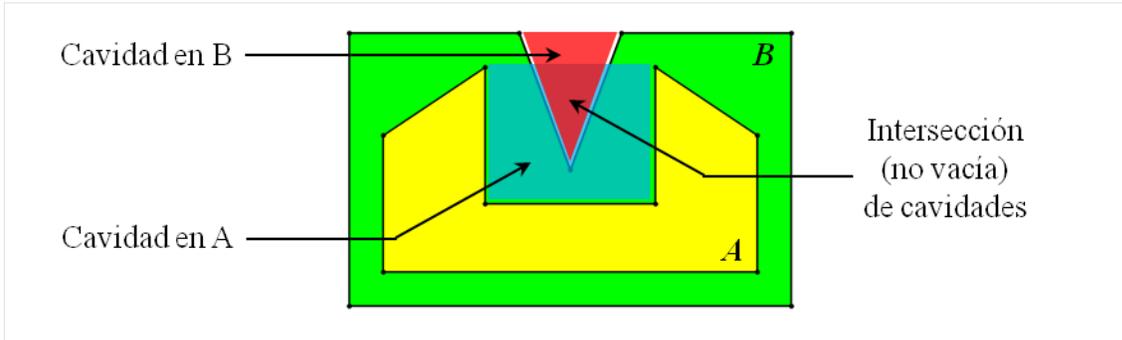


Figura 2.7: Cavidades con intersección no vacía y formación de nuevos polígonos.

**Teorema 2.5.3** *Todos los vértices de la cubierta convexa de  $I$ , de la subtrayectoria entre  $p_s$  y  $p_e$  son vértices de la cubierta convexa relativa  $CH_B(A)$  [7, p. 118].*

Tal como en el proceso descrito previamente, si el polígono  $I$  tiene una cavidad  $CAV_i(I)$  con vértices de  $O$  dentro de ella, entonces los vértices de la cubierta de  $CAV_i(I)$  y los vértices  $O$  en el interior de  $CAV_i(I)$  constituyen a su vez un nuevo polígono interior, y los vértices que forman  $CAV_i(I)$  incluyendo los de su cubierta, constituyen un nuevo polígono exterior. Esto lleva a la idea básica de un algoritmo recursivo para el cálculo de la cubierta convexa relativa.

## 2.5.2. Descripción detallada

El algoritmo tiene como datos de entrada una secuencia de los  $n$  vértices de un polígono simple interior  $A = \langle p_1, p_2, \dots, p_n \rangle$ , y una secuencia de los  $m$  vértices de un polígono simple exterior  $B = \langle q_1, q_2, \dots, q_m \rangle$ . Tales polígonos  $A$  y  $B$  cumplen que  $A \subseteq B$ . Como dato de salida se obtiene una lista tipo *deque* constituida por los vértices de la cubierta convexa relativa  $CH_B(A)$ .

Como paso inicial se obtiene la cubierta convexa del polígono interior  $A$  y la del polígono exterior  $B$  utilizando el algoritmo de Melkman. El resultado de estas operaciones es guardado en las listas  $D(I)$  y  $D(O)$ , respectivamente.

Después se revisa cada par consecutivo de elementos de  $D(I)$  hasta que se detecta la existencia de una cavidad. Se propone realizar esta operación buscando una diferencia mayor a uno en los índices de cada par de vértices en  $D(I)$ . Es decir, para dos vértices consecutivos  $p_j$  y  $p_i$

en  $D(I)$ , se tiene una cavidad si  $(i - j) > 1$ . Cuando se tiene una cavidad, al vértice inicial  $p_j$  y al vértice final  $p_i$  se les denota como  $p_s$  y  $p_e$ , respectivamente. Estos mismos vértices forman a su vez la cubierta de  $CAV_i(A)$ .

Si no existe cavidad alguna  $CAV_i(A)$  en  $D(I)$ , significa que la lista  $D(I)$  ya contiene todos los vértices que constituyen la frontera de  $CH_B(A)$ .

Si se detecta una cavidad  $CAV_i(A)$  en  $D(I)$ , se busca alguna cavidad  $CAV_j(B)$  en  $D(O)$ , si existiera, que tenga al menos uno de sus vértices en el interior de  $CAV_i(A)$ .

Cuando esto ocurre, es posible definir dos nuevos polígonos, uno interior y otro exterior. El nuevo polígono interior  $I$ , queda constituido por todos los vértices de  $B$  dentro de  $CAV_i(A)$ , además del vértice inicial y final de  $CAV_i(A)$ . El nuevo polígono exterior  $O$  se forma con todos los vértices de  $CAV_i(A)$ , incluyendo sus vértices inicial y final. Nótese que el segmento de recta que forma la cubierta de  $CAV_i(A)$  es parte de ambos polígonos nuevos.

Cuando solo existe un vértice de  $CAV_j(B)$  en el interior de  $CAV_i(A)$ , el polígono  $I$  estará conformado únicamente por tres vértices. Es bien sabido que la cubierta convexa de un polígono triangular es el polígono mismo, por lo tanto, se tendrá que  $CH(I) = I$ . Esto implica que  $I$  no tiene cavidades, y por consiguiente no se generarán nuevos polígonos interiores y exteriores dentro de sus cavidades. Esta es la *base* de la recursión. Cuando se tiene esta situación, solamente se agrega el vértice de  $CAV_j(B)$  entre los vértices de la cubierta de  $CAV_i(A)$  localizados dentro de la lista  $D(I)$ .

Cuando  $I$  tiene más de tres vértices se pueden presentar tres casos:

1. Si  $I$  es convexo, entonces  $CH(I) = I$  y todos los vértices de  $I$  son parte de  $CH_B(A)$ , y se agregan entre los vértices de la cubierta de  $CAV_i(A)$  en  $D(I)$ .
2. Si  $I$  no es convexo pero no tiene vértices de  $O$  dentro de alguna cavidad  $CAV_k(I)$ , entonces solo se obtiene  $CH(I)$ , y sus vértices se agregan entre los vértices de la cubierta de  $CAV_i(A)$  dentro de la lista  $D(I)$ .
3. Si  $I$  no es convexo y tiene vértices de  $O$  dentro de alguna cavidad  $CAV_k(I)$ , entonces se repite el proceso inicial, formando un nuevo polígono interior y uno exterior de la misma forma que se construyeron  $I$  y  $O$  con respecto de  $A$  y  $B$ . Este proceso se repite de forma recursiva, disminuyendo el tamaño (número de vértices) de los polígonos interiores y exteriores hasta que se llega al caso base.

Al número de ocasiones en que se realiza el procedimiento recursivo dentro de una cavidad anidada se le denomina *profundidad*. Cuando esta profundidad es mayor a uno se tienen las llamadas *cavidades anidadas* (Figura 2.8).

En las descripciones previas, cuando se dice que “*todos los vértices de  $CH(I)$  se agregan entre los vértices de la cubierta de  $CAV_i(A)$  en  $D(I)$* ”, debe entenderse que se agregan todos los vértices de  $CH(I)$  excepto los vértices de la cubierta de  $CAV_i(A)$ , puesto que estos ya existen en  $D(I)$ .

En [2], la autora hace las siguientes observaciones sobre la complejidad en tiempo del algoritmo:

- Cada vértice en  $A$  o  $B$  es accesado por el algoritmo tantas veces como el número de ocasiones que se realiza el procedimiento recursivo. Esto define al algoritmo de complejidad

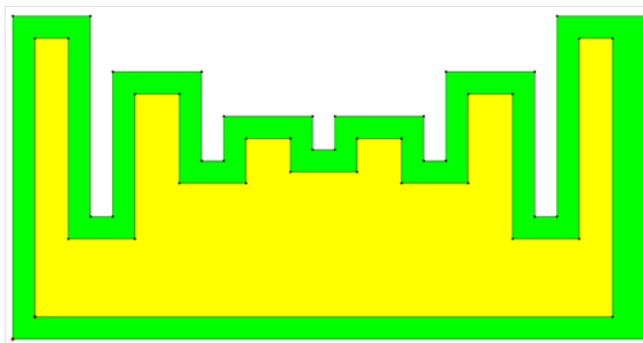


Figura 2.8: Cavidades anidadas.

lineal, medida en el número total de vértices de  $A$  y  $B$ , si el número de pasos recursivos a ser efectuados en cada vértice está acotado por una constante. Sin embargo, el algoritmo es de tiempo cuadrático en el peor caso.

- El peor caso en complejidad en tiempo se presenta cuando hay un número pequeño de cavidades grandes en  $A$  y  $B$ , causando repetidas llamadas recursivas.
- Para el caso general, si existen muchas cavidades en  $A$ , intuitivamente varias de ellas serán pequeñas, ya que entre más cavidades existan, tenderán a estar constituidas por un número pequeño de vértices. En esta situación, el algoritmo tiene un comportamiento lineal.
- Si hay solo una cavidad grande, entonces la recursión solo se ejecuta en esta única cavidad, y después de cada paso recursivo se tiene básicamente la misma cota superior que en el paso previo, pero para un número menor de vértices. Cabe recordar que entre cada paso se realizan diversas operaciones con complejidad lineal, tales como la determinación de la cubierta convexa de los nuevos polígonos formados en cada paso recursivo.

---

---

**Entrada:** Dos polígonos simples  $A = \langle p_1, p_2, \dots, p_n \rangle$  y  $B = \langle q_1, q_2, \dots, q_m \rangle$ , tales que  $A \subseteq B$ .

**Salida:** La cubierta convexa relativa en  $D(I)$ .

1: Inicializar  $D(I) = \emptyset$  y  $D(O) = \emptyset$ .

2: Llamar la función  $\text{RCH}(A, B, D(I), D(O), n, m, p_1, p_n)$

---

**Función  $\text{RCH}(A, B, D(I), D(O), n, m, p_1, p_n)$**

---

1: Calcular la cubierta convexa de  $I$  y  $O$  en *deques*  $D(I)$  y  $D(O)$ , respectivamente;  $l$  es el número de vértices en  $I$  y  $t$  el número de vértices de  $O$ ;  $p_s$  es el vértice inicial y  $p_e$  el vértice final del polígono interior;  $k = 1$  y  $j = 1$  son variables de ciclo.

2: Remover el elemento final de cada *deque*  $D(I)$  y  $D(O)$ .

3: **while**  $k < l$  **do**

4:     **if** Existe una cavidad entre dos vértices consecutivos  $v_k$  y  $v_{k+1}$  en  $D(I)$  **then**

5:          $p_s = v_k, \quad p_e = v_{k+1}$ .

6:         **while**  $j < t$  **do**

7:             **if** Se traslapa una cavidad entre dos vértices consecutivos de  $D(O)$  **then**

8:                 Actualizar  $I$  de tal forma que incluya  $p_s$  y  $p_e$ , y todos los vértices de  $O$  dentro de la cavidad de  $I$ ;  $L$  es el número de vértices.

9:                 **if**  $L > 3$  **then**

10:                     Actualizar  $O$  de tal forma que incluya  $p_s$  y  $p_e$ , y todos los vértices en  $I$  dentro de la cavidad de  $I$ ;  $T$  es el número de vértices.

11:                     Llamar la función  $\text{RCH}(I, O, D(I), D(O), L, T, p_s, p_e)$ .

12:                 **end if**

13:                 Insertar  $q$  entre  $p_s$  y  $p_e$  en  $D(I)$ .

14:             **end if**

15:         **end while**

16:         **return**  $D(I)$

17:     **end if**

18: **end while**

19: **return**  $D(I)$

---

Figura 2.9: Pseudocódigo del algoritmo de G. Klette [2].



# Capítulo 3

## Propuesta de algoritmo

La motivación para desarrollar un nuevo algoritmo para determinar la cubierta convexa relativa para polígonos simples arbitrarios surgió de un cuidadoso análisis del algoritmo de G. Klette publicado en [2]. Como resultado de este análisis, y del trabajo de programación realizado para su implementación, se concluyó que este es erróneo. Se trabajó en la corrección de los errores identificados, lo cual llevó a la obtención de una nueva versión del algoritmo. En este capítulo se presenta tanto el análisis del algoritmo de G. Klette, como el desarrollo detallado de la versión corregida.

### 3.1. Análisis del algoritmo de G. Klette

En el año 2011, Gisela Klette presentó en la *International Conference on Digital Geometry for Computer Imagery (DGCI)* celebrada en Nancy, Francia, una propuesta de un algoritmo para resolver el problema de la cubierta convexa relativa para dos polígono simples en el plano. La característica más importante de tal algoritmo es el uso de la recursión.

En la Sección 2.5 del Capítulo 2 se explicó a detalle el funcionamiento de este algoritmo y se presentó su pseudocódigo de acuerdo a la publicación [2]. En esta sección se realiza un análisis del algoritmo y se explican algunos casos en los que este no obtiene los resultados esperados.

#### 3.1.1. Observaciones

A continuación se enlistan diversas observaciones hechas al pseudocódigo del algoritmo de G. Klette, basados en las ideas y procedimientos descritos en la publicación [2]. Dicho pseudocódigo también es reportado al final de la Sección 2.5 de esta tesis, en la página 35. Tales observaciones son:

- En el pseudocódigo se asume que las polilíneas de entrada inician en un vértice extremo (con coordenada  $y$  mínima). Para que estas polilíneas inicien como lo requiere el algoritmo, en general, sería necesario algún tipo de preprocesamiento.
- En la línea 1 de la Función RCH del pseudocódigo se dice que deben obtenerse la cubierta convexa de  $A$  y de  $B$  mediante el algoritmo de Melkman. En [2], después de presentar el pseudocódigo se menciona lo siguiente:

« El algoritmo de Melkman entrega la cubierta convexa en una *deque* con el primer elemento al final de la *deque* y también al inicio de la *deque*. »

De igual forma, cuando se describe el ejemplo de la Figura 3 del texto [2], dice:

« En este caso, la cubierta convexa de  $A$  es guardada en una *deque* y el conjunto de vértices es igual a  $D(A) = \langle p_1, p_2, p_{10}, \dots, p_{16}, p_1 \rangle$ . ».

Estas dos afirmaciones anteriores son falsas, ya que el resultado del algoritmo de Melkman, si bien es una *deque* con el primer elemento igual al último, este elemento no corresponde al primer vértice de  $A$ . Este elemento inicial y final corresponde al vértice final de la polilínea que integra la frontera de la cubierta convexa.

Esto es fácil de observar analizando el pseudocódigo del algoritmo de Melkman, ya sea en la fuente original [8] (Apéndice C) o en su versión “equivalente” presentada en [7]. En ellos se puede ver que el algoritmo analiza un vértice del polígono de entrada en cada iteración. Este vértice se agrega al inicio y al final de la *deque* de salida solo si se confirma que no se encuentra ya en el interior de la cubierta convexa calculada hasta un ciclo previo. Esto indica que el último vértice agregado a la *deque* nunca podría ser el primer elemento de  $A$ .

Cabe mencionar que este resultado es independiente del sentido de trazado de los vértices de entrada (Figura 3.1). En la fuente original [8] el algoritmo de Melkman opera con una polilínea en sentido antihorario, y en [7] se presenta una versión para una entrada en sentido horario. Es importante destacar que la versión del pseudocódigo presentada en [7] es una versión reducida del algoritmo original, la cual funciona únicamente si la primera tripleta de puntos forma una vuelta derecha. El pseudocódigo original en [8] es ligeramente más extenso pero no tiene esta restricción. En la Sección 4.2 se detallan estas diferencias y sus implicaciones.

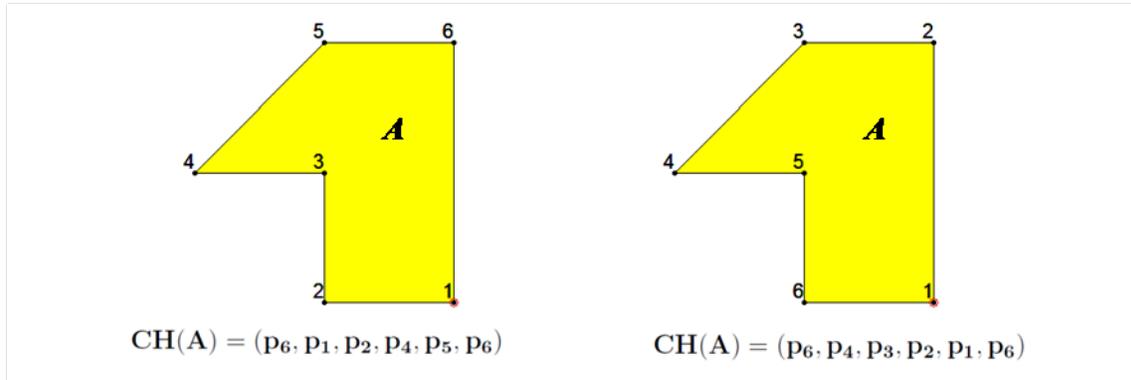


Figura 3.1: Resultados del algoritmo de Melkman para entradas en sentido horario (izquierda) y en sentido antihorario (derecha).

- En la línea 2 de la Función RCH se dice que debe eliminarse el último elemento de  $D(I)$  y  $D(O)$ . Dado que la *deque* representa una polilínea cerrada, al eliminar el vértice final se elimina la información del segmento que une el último con el primer vértice de la cubierta

convexa. De acuerdo al ejemplo en [2] antes citado, se eliminaría el segmento  $\overline{p_{16}p_1}$ . Esta situación es importante ya que al eliminarse no es posible detectar alguna cavidad entre estos dos vértices.

Incluso teniendo el resultado correcto del algoritmo de Melkman, es decir, para el caso en que  $D(A) = \langle p_{16}, p_1, p_2, p_{10}, \dots, p_{16} \rangle$ , no tendría sentido eliminar el primer vértice de la *deque*, por el motivo expuesto anteriormente.

- En la línea 3 se inicia una operación iterativa en la cual se tiene al número de elementos del polígono  $I$  ( $l$ ) como condición de término. Hacer esto es un error de acuerdo a lo siguiente.

De acuerdo a las líneas subsecuentes del pseudocódigo, este número de iteraciones tienen como objetivo verificar pares de vértices en  $D(I)$  en busca de cavidades. Por ello, lo conveniente es utilizar en lugar de  $l$ , una variable, llámese  $s$ , definida como el número de elementos de  $D(I)$  sin eliminar su último vértice (o su primer vértice en el caso corregido).

Si se realiza la operación  $(k + 1) \bmod l$  se comenzará un nuevo ciclo en el que se buscarán cavidades en segmentos ya revisados. Si se utiliza la variable  $l$ , en la línea 4 llegará el momento en el que no exista un vértice  $d_{k+1}$  y resultará en un error puesto que, en general, un polígono  $I$  tiene más vértices que su cubierta convexa.

Por otra parte, cabe la posibilidad de que al ir agregando vértices en  $D(I)$  en cada cavidad de  $A$ , llegue el momento en el que el número de elementos de  $D(I)$  sea igual o mayor a  $l$ . En esta situación, la condición  $k < l$  será una limitante y las iteraciones finalizarán antes de revisar todos los segmentos de  $D(I)$  en búsqueda de cavidades.

La *deque*  $D(I)$  tiene  $s$  elementos que corresponden a  $s - 1$  vértices de la cubierta convexa de  $I$ . Por lo tanto, al sustituir en la línea 3 la condición  $k < l$  por  $k < s$  es posible verificar solo los  $s - 1$  segmentos  $\overline{d_k d_{k+1}}$  que constituyen  $D(I)$  en busca de cavidades.

Considerando que el número de elementos de  $D(I)$  aumenta al agregarse vértices de cavidades de los polígonos  $A$  y  $B$  con intersección no vacía, el algoritmo debe ajustar el valor de  $s$  de acuerdo al número de vértices agregados para que no ocurra el problema anterior.

- La línea 4 prevé la detección de cavidades entre dos vértices consecutivos de  $D(I)$ . Aunque el pseudocódigo no describe algún método de detección, en el texto se menciona que esto se realiza revisando la diferencia entre los índices de dos vértices consecutivos de  $D(I)$ , expresándolo de la siguiente forma:

« Una cavidad en  $A$  con el vértice inicial  $p_s = p_j$  y el vértice final  $p_e = p_i$  ha sido encontrada si  $(i - j) > 1$ . »

Esto es válido para todos los pares de vértices consecutivos de  $D(I)$ , excepto para el último par (o para el primer par en el caso del resultado corregido).

Este par es el que forma el segmento entre el último y el primer vértice de la cubierta convexa, cerrando la polilínea para convertir la *deque* actual en la representación de la frontera de un polígono. La diferencia entre los índices de estos dos vértices siempre será mayor a uno, sin embargo esto no indica que exista una cavidad entre ellos. Para

detectar una cavidad en esta situación, es indispensable que no se elimine el último vértice de  $D(A)$  en la línea 2.

Posteriormente, para descartar la existencia de una cavidad entre este par de vértices, solo es necesario verificar que los índices del último y primer vértice de la cubierta convexa sean el último y primer vértice del polígono en cuestión. Otra opción es utilizar la operación módulo entre los índices y verificar con la condición citada previamente.

Otro problema con el método propuesto para la detección de cavidades es cuando se tiene un polígono dado por  $I = \langle p_2, q_3, q_4, \dots, q_{10}, q_{11}, p_{10} \rangle$  (obtenido del ejemplo de la Figura 3 en [2]). En este caso el vértice inicial y final son parte del polígono interior  $A$  y los restantes son del polígono exterior  $B$ . Como puede observarse, el primer y último segmento del polígono  $I$  se forman con vértices de polígonos diferentes, por lo cual sus índices no serán, en general, consecutivos. Esto permite que incluso cuando la diferencia entre sus índices sea mayor a uno, no exista cavidad alguna entre ellos. En contraste, si los índices son consecutivos, o incluso iguales, es posible que exista una cavidad entre ellos.

- En la línea 6, la condición  $j < t$  debería ser sustituida por  $j < r$ , donde  $r$  es el número de elementos de  $D(O)$ . La razón es análoga a la expuesta previamente para la línea 3.
- Por los motivos expuestos anteriormente, los valores  $l$ ,  $t$  y  $T$  son innecesarios.
- En la línea 8, se debe distinguir entre la notación de  $I$  e  $I_{new}$ , ya que puede ser confuso referirse a los dos polígonos con la misma notación en la misma frase.
- En la línea 10, se hace referencia a un polígono  $I$  inexistente, dado que fue redefinido en la línea 8. Resolviendo la ambigüedad de la línea 8 también se resuelve este problema.
- La línea 13 hace referencia a la base de la recursión. En ella se menciona que debe agregarse el vértice  $q$  entre los vértices  $p_s$  y  $p_e$  de  $D(I)$ , lo cual no es permitido en las operaciones de una *deque*, puesto que en ella únicamente pueden agregarse elementos al inicio o al final. Al permitirse agregar un elemento en una posición específica, la *deque*  $D(I)$  deja de serlo, convirtiéndose en solo una lista.
- Aunque en la metodología descrita en el texto de [2] se explica de forma clara, en su pseudocódigo no existe una línea en la que se haga explícita la inserción de una  $D(I)$  en su posición correspondiente en su paso recursivo previo.

### 3.1.2. Fallas en el algoritmo

Incluso aplicando las correcciones necesarias de acuerdo a las observaciones presentadas en la subsección anterior, existen dos situaciones bien definidas en las que el algoritmo no es capaz de obtener el resultado correcto.

#### Caso 1:

El primero de estos casos de falla se presenta durante la redefinición de los polígonos interior y exterior después de la primera recursión. Cuando se llega a esta etapa, se generan dos zonas ciegas para el algoritmo en cada cavidad por cada paso adicional de recursión. Estas zonas ciegas

pueden incluir vértices que pertenecen a la cubierta convexa relativa, y por tanto su omisión puede afectar directamente al resultado total.

Un aspecto muy importante de esta falla es que tiene un origen geométrico, y no depende del proceso de recursión.

La Figura 3.2 muestra una configuración similar a la del ejemplo de la Figura 3 en [2]. Este ejemplo es muy útil para mostrar las zonas que el algoritmo no puede revisar y las consecuencias que puede tener esto en el resultado final.

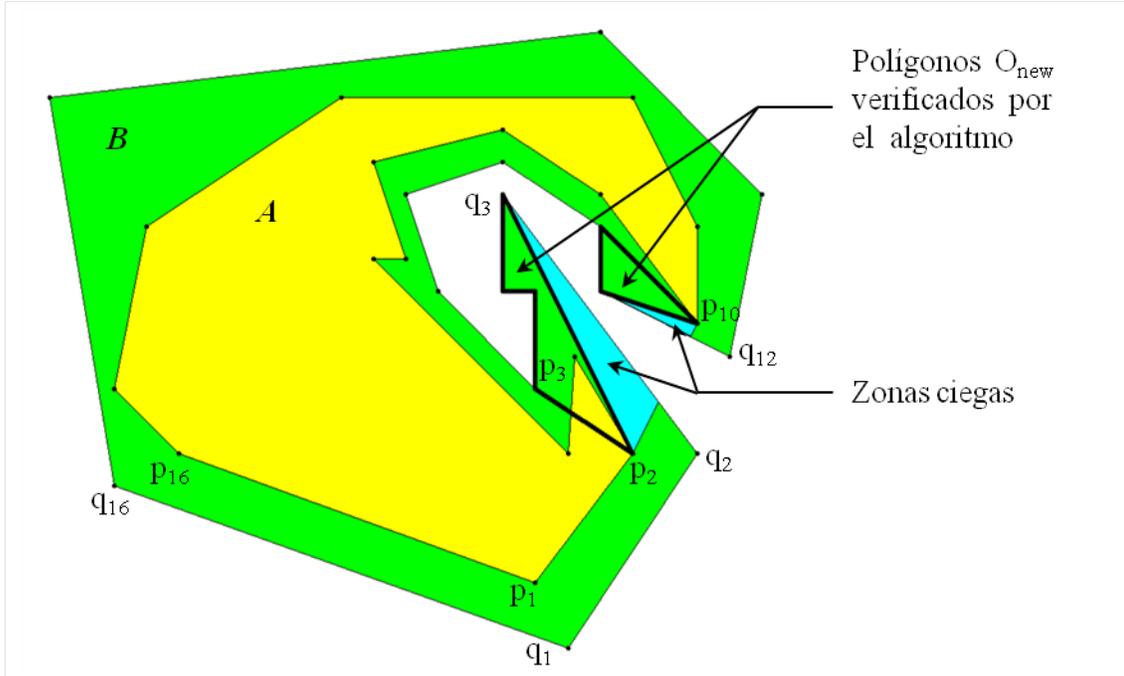


Figura 3.2: Zonas ciegas en el ejemplo de G. Klette.

Aplicando el algoritmo de G. Klette al ejemplo de la Figura 3.2, se tiene que existe una cavidad de  $A$  que tiene una intersección no vacía con una cavidad de  $B$ . Entonces se define el polígono exterior  $O = \langle p_2, p_3, \dots, p_{10} \rangle$  y un polígono interior  $I = \langle p_2, q_3, q_4, \dots, q_{10}, q_{11}, p_{10} \rangle$ . En el siguiente paso de recursión se calcula la *deque*  $D(I) = \langle p_2, q_6, \dots, q_{10}, p_{10} \rangle$  y se buscan cavidades en ella. Posteriormente se define  $O_{new} = \langle p_2, q_3, \dots, q_6 \rangle$ , e  $I_{new} = \langle p_2, p_3, q_6 \rangle$ .

Sin embargo  $O_{new}$  no cubre toda el área que debe verificarse para buscar vértices de  $A$ . Suponiendo que se detecta correctamente una cavidad entre los segmentos  $\overline{p_2q_6}$  y  $\overline{p_{10}q_{10}}$ , la primer zona ciega para el algoritmo es la que se forma entre los vértices  $p_2$ ,  $q_3$ , y el vértice que en adelante se denominará *inter<sub>S</sub>*, el cual resulta de la intersección del segmento  $\overline{q_2q_3}$  y la cubierta  $\overline{p_2p_{10}}$ , es decir, esta zona ciega puede definirse como  $Z_S = \langle p_2, inter_S, q_3 \rangle$ . De forma ideal  $O_{new}$  debería definirse como  $O_{new} = \langle p_2, inter_S, q_3, \dots, q_6 \rangle$ .

Si siguiendo el algoritmo descrito en [2], si  $p_3$  se encontrara a la derecha del segmento  $\overline{p_2q_3}$  (Figura 3.3), nunca sería detectado y por tanto no sería agregado al resultado final aun cuando debería formar parte de él.

De forma análoga, se forma una zona ciega al final de la cavidad. Definiendo como *inter<sub>E</sub>* al vértice que resulta de la intersección del segmento  $\overline{q_{11}q_{12}}$  y la cubierta  $\overline{p_2p_{10}}$ . Esta segunda zona

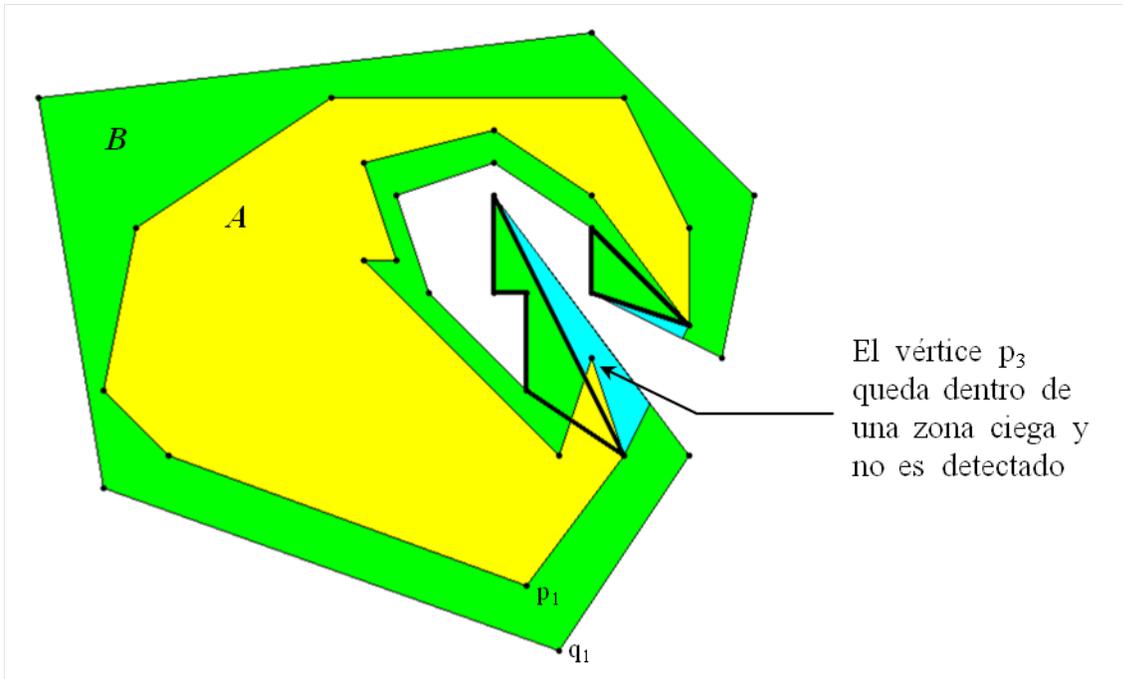


Figura 3.3: El vértice  $p_3$  que forma parte del resultado final no es detectado por el algoritmo.

ciega puede definirse como  $Z_E = \langle p_{10}, inter_E, q_{11} \rangle$ . Aquí, el polígono  $O_{new}$  el algoritmo lo define como  $O_{new} = \langle q_{10}, q_{11}, p_{10} \rangle$ , cuando debería ser  $O_{new} = \langle q_{10}, q_{11}, inter_E, p_{10} \rangle$ .

Aun cuando el algoritmo de G. Klette se propone para el caso general, esta falla también se presenta en el caso de polígonos isotéticos (Figura 3.4). Por esta razón, es falso cuando en [2] se indica que:

« El nuevo algoritmo es para el caso general de polígonos simples (como en el escenario robótico), pero también para polígonos con más restricciones, como en el caso de imágenes digitales ».

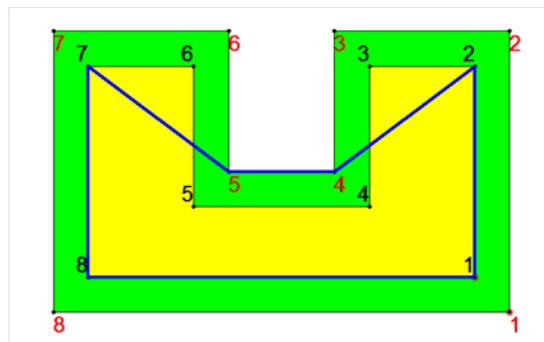


Figura 3.4: Falla del algoritmo de G. Klette en el caso de polígonos isotéticos.

### Caso 3:

Existe una segunda situación especial que el algoritmo no contempla y, por lo tanto, es posible que no obtenga el resultado correcto. Este caso ocurre cuando al menos un vértice del polígono exterior es colineal con algún lado (o vértice) de la frontera de una cavidad del polígono interior, excepto cuando se trata de su cubierta (Figura 3.5).

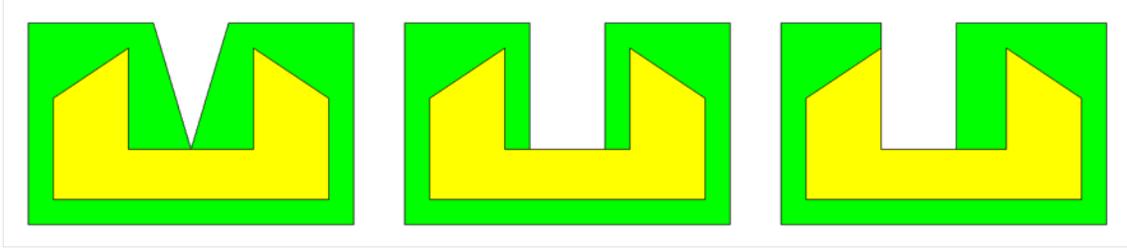


Figura 3.5: Casos de colinealidades no contempladas por el algoritmo de G. Klette.

El error del algoritmo consiste en que este solo busca vértices del polígono exterior en el interior de la cavidad, pero no en su frontera. El problema de no detectar estos casos es que estos vértices del polígono exterior frecuentemente forman parte de la cubierta convexa relativa.

En la Figura 3.5 se muestra que no solo pueden ser vértices, sino también partes de lados o lados completos del polígono exterior los que pueden ser colineales con lados de la frontera de una cavidad del polígono interior. En los ejemplos mostrados se hacen evidentes las consecuencias de omitir la detección de estos casos.

## 3.2. Versión revisada del algoritmo

Una vez que se analizaron las observaciones y las fallas descritas en la sección anterior se decidió desarrollar una versión revisada del algoritmo de G. Klette. Sin embargo, en el transcurso del proceso de corrección, la estructura de esta nueva versión se modificó de tal forma que la nueva propuesta de algoritmo se basa en iteraciones, no en recursión.

Con este enfoque iterativo se eliminaron los obstáculos que presentaba la recursión, sobre todo para la etapa de implementación. También, al realizar diversas modificaciones e incluir algunos conceptos nuevos, muchas de las observaciones hechas al algoritmo de G. Klette no aplican o fueron subsanadas.

En los siguientes apartados se explica a detalle el algoritmo propuesto y se presenta su pseudocódigo.

### 3.2.1. Descripción detallada

Esta nueva versión está basada en gran parte en las definiciones y la notación presentadas en [2], sin embargo el procedimiento tiene un desarrollo diferente, el cual se detalla a continuación.

Como datos de entrada se tienen dos polígonos simples, cada uno dado por la secuencia ordenada de sus vértices, un polígono interior  $A = \langle p_1, p_2, \dots, p_n \rangle$  y un polígono exterior  $B = \langle q_1, q_2, \dots, q_m \rangle$ , tales que cumplen la condición  $A \subseteq B$ . Se supone que ambas secuencias de vértices corresponden a un sentido horario de trazado. Como dato de salida se obtienen los vértices de la cubierta convexa de  $A$  relativa a  $B$  guardados en la lista  $CH(A)$  que sigue

correspondiendo a un trazado en sentido horario. La Figura 3.6 muestra los polígonos  $A$  y  $B$  que servirán de ejemplo para la descripción del algoritmo.

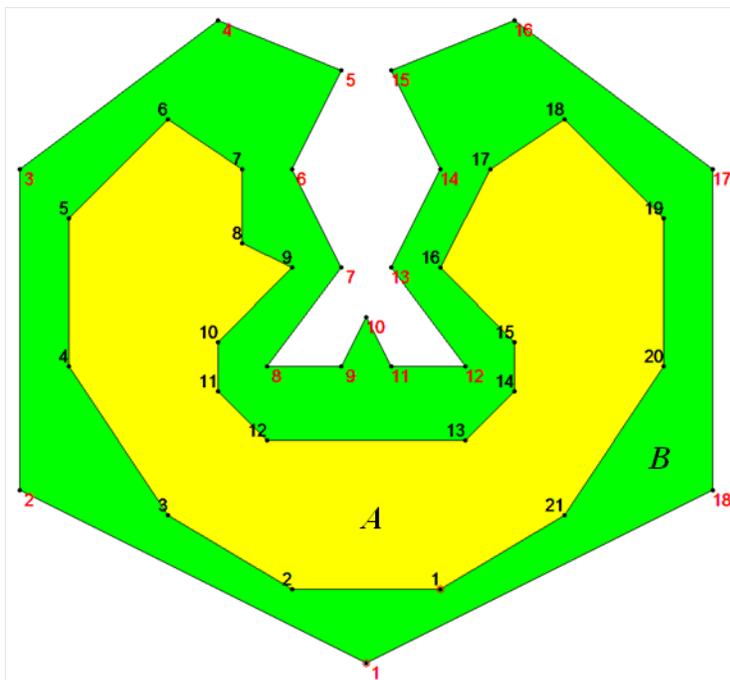


Figura 3.6: Polígonos interior y exterior de ejemplo.

Se supone que el primer vértice de  $A$  y de  $B$ , ambos son vértices extremos. De esta forma se asegura que el procedimiento comienza en un vértice que pertenece a la cubierta convexa relativa.

Esta condición es fácil de lograr aplicando un sencillo preprocesamiento a las polilíneas de entrada, las cuales para dos polígonos arbitrarios, en general, no comienzan en un punto extremo. Una opción simple para realizar este preprocesamiento es hacer una búsqueda rápida en el polígono  $A$ , en la lista de una de las coordenadas ( $x$  o  $y$ ) para encontrar el valor máximo (o mínimo). En [2] se utiliza el vértice con  $y$  mínima. Una vez que se ha identificado el vértice extremo, se hace un corrimiento cíclico de índices en  $A$ , de tal forma que ahora este vértice extremo tenga un índice de 1, y de esta forma se considere como vértice inicial.

Una vez que la suposición previa ha sido satisfecha, se determina la cubierta convexa de los polígonos  $A$  y  $B$  utilizando el algoritmo de Melkman. Ambos resultados se guardan en las *deques*  $CH(A)$  y  $CH(B)$ , respectivamente. Estas *deques* resultantes tienen en sus dos extremos el mismo vértice, que corresponde al último vértice procesado de  $A$  (o de  $B$ ) que es parte de su cubierta convexa  $CH(A)$  (o  $CH(B)$ ). En el ejemplo de la Figura 3.6,  $CH(A) = \langle p_{21}, p_1, p_2, \dots, p_5, p_6, p_{18}, p_{19}, p_{20}, p_{21} \rangle$ . Este resultado se modifica eliminando su primer elemento, y repitiendo el segundo al final con un índice  $n + 1$ , es decir,  $p_1 = p_{n+1}$ . Así, se obtiene  $CH(A) = \langle p_1, p_2, p_3, \dots, p_5, p_6, p_{18}, p_{19}, p_{20}, p_{21}, p_{22} \rangle$ .

En la *deque* de la secuencia de vertices de cada polígono  $A$  y  $B$  también se repite el primer vértice al final, con un índice  $n + 1$ . Estas operaciones se realizan con el objetivo de formar *deques* en las cuales puedan realizarse operaciones con el segmento que une al último con el

primer vértice sin que se produzcan errores con sus índices. En el caso de  $CH(A)$ , esto permite detectar cavidades en este segmento sin dificultades.

Como se ha mencionado antes, a diferencia del algoritmo de G. Klette, esta nueva versión se basa en la iteración. Cada ciclo verifica uno de los  $s$  segmentos que componen  $CH(A)$ . El algoritmo busca cavidades en  $A$  mediante la diferencia de índices en dos vértices consecutivos de  $CH(A)$ , tal como se describe en [2] y en la Subsección 2.5.2 de esta tesis.

Una vez que ha sido detectada una cavidad en  $A$ , denotada como  $CAV(A)$ , se genera un nuevo polígono exterior  $O$  formado por todos los vértices de  $A$ , desde el vértice  $CH(A)_i$  hasta el  $CH(A)_{i+1}$ . De acuerdo a la Figura 3.6,  $O = \langle p_6, p_7, p_8, \dots, p_{17}, p_{18} \rangle$ .

Posteriormente se busca una cavidad en  $B$  tal que tenga una intersección no vacía con la región  $R(O)$ , la cual se define como  $R(O) = O \setminus \overline{CH(A)_i CH(A)_{i+1}}$ , es decir, el polígono  $O$  sin la cubierta de la cavidad  $CAV(A)$ .

Cuando se encuentra una cavidad  $CAV(B)$  que cumpla esta condición (si existe), se genera entonces un nuevo polígono interior  $I$  formado por todos los vértices de  $CAV(B)$  dentro de  $O$  añadidos a los vértices  $CH(A)_{i+1}$  y  $CH(A)_i$ . Según la Figura 3.6,  $I = \langle p_{18}, p_6, q_6, q_7, \dots, q_{13}, q_{14} \rangle$ .

Mediante el algoritmo de Melkman se obtiene la cubierta convexa del nuevo polígono  $I$ . Considerando que  $I$  está definido en sentido antihorario, se obtiene  $CH(I) = \langle q_{12}, q_8, p_6, p_{18}, q_{12} \rangle$  que se compone de  $S$  elementos. Sin embargo, uno de estos elementos ( $q_{12}$ ) se presenta al inicio y al final, por lo cual el número efectivo de vértices que forman  $CH(I)$  es  $S - 1$ . Si a estos  $S - 1$  elementos se eliminan los dos vértices que comparte con  $O$ , es decir,  $CH(A)_i = p_6$  y  $CH(A)_{i+1} = p_{18}$  se obtienen solo  $S - 3$  vértices. Estos  $S - 3$  elementos son parte de la cubierta convexa relativa y se agregan a ella entre  $CH(A)_i = p_6$  y  $CH(A)_{i+1} = p_{18}$  en  $CH(A)$ . Al número de vértices de  $CAV(B)$  que se agregan a  $CH(A)$ , se le denomina  $u$ , es decir  $u = S - 3$ .

Puesto que se agregan  $u$  vértices a  $CH(A)$ , también se agregan  $u$  segmentos. Cada uno de estos segmentos debe verificarse para revisar posibles nuevas cavidades entre ellos. Por esta razón es necesario actualizar el número de ciclos en la iteración principal agregando  $u$  unidades al número  $s$  de ciclos previstos inicialmente.

Hasta este punto, el algoritmo es muy similar al propuesto anteriormente por G. Klette. Sin embargo, en lo siguiente se comienzan a mostrar las diferencias sustanciales. Para detallar estas diferencias es necesario definir algunos nuevos conceptos, ilustrados en la Figura 3.7.

Cuando se obtiene el polígono  $CH(I)$ , se generan dos nuevos segmentos muy importantes para esta nueva versión. Estos dos segmentos son los que conectan el polígono interior con el exterior, uno a cada extremo de la cavidad.

En esta versión del algoritmo, se le llamará *cubierta principal* a lo que en la definición 1.1.7 se presentó como *cubierta*. De la misma manera, se le llamará *cavidad principal* a la cavidad cuya cubierta es la *cubierta principal*. También, cuando se tiene una cavidad en  $A$ , con vértices de  $B$  al interior (con  $A$  y  $B$  en sentido horario) se define lo siguiente:

**Definición 3.2.1** *En un polígono  $CH(I)$  (formado de acuerdo a la metodología previamente descrita), al segmento que inicia en el vértice de índice más pequeño de la cubierta principal, y termina en el vértice de índice más pequeño de  $CH(I)$  ubicado en el interior topológico de la cavidad principal, se le denomina **cubierta secundaria inicial**. A su vez, al segmento que inicia en el vértice de índice más grande de  $CH(I)$  ubicado en el interior topológico de la cavidad principal, y termina en el vértice de índice más grande de la cubierta principal, se le denomina **cubierta secundaria final**.*

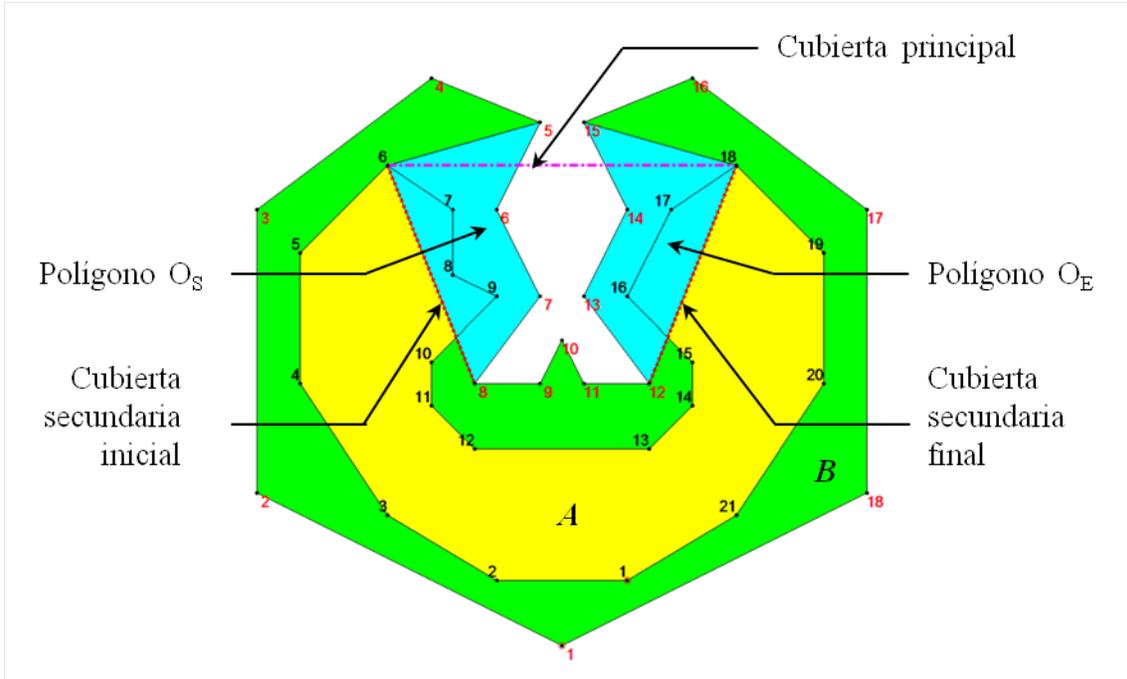


Figura 3.7: Cubierta principal y cubiertas secundarias.

Estas cubiertas secundarias tienen la característica de que inician en un vértice del polígono interior  $A$  y terminan en un vértice del polígono exterior  $B$  (o viceversa). El calificativo de *inicial* y *final* lo determinan solo los índices de los vértices que conforman la cubierta principal.

De acuerdo a la Figura 3.7, la cubierta principal es el segmento  $\overline{p_6 p_{18}}$ . La cubierta secundaria inicial es el segmento  $\overline{p_6 q_8}$  y la cubierta secundaria final es  $\overline{p_{18} q_{12}}$ .

A pesar de que estos dos últimos segmentos no son propiamente *cubiertas* de acuerdo a la definición 1.1.7, pueden considerarse análogas a este concepto dado que en el algoritmo tienen una función similar a la cubierta principal de la cavidad.

A partir de cada una de estas cubiertas secundarias se generan nuevos polígonos interior y exterior, llamados  $I_S$  y  $O_S$  de forma correspondiente partiendo de la cubierta secundaria inicial, y los polígonos  $I_E$  y  $O_E$  partiendo de la cubierta secundaria final. La formación de estos nuevos polígonos tiene como objetivo verificar las zonas que son ciegas para el algoritmo de G. Klette.

El polígono  $O_S$  se forma en sentido antihorario, iniciando en la cubierta secundaria inicial formada por los vértices  $CH(A)_i$  y  $CH(A)_{i+1}$ . Hay que recordar que el vértice  $CH(A)_{i+1}$  cambió al agregarse los  $u$  vértices de  $CH(I)$  a  $CH(A)$ . Por este motivo, en este punto del algoritmo  $CH(A)_{i+1}$  es ahora el primer vértice de  $CH(I)$  dentro del polígono  $O$  ( $q_8$ ). Después se agregan todos los vértices de  $B$  previos a  $CH(A)_{i+1}$  finalizando con el primer vértice de  $B$  fuera de la cavidad principal o colineal con su cubierta.

El polígono  $I_S$  se forma iniciando con la cubierta secundaria inicial, y agregando todos los vértices de  $A$  dentro de  $O_S$ , o colineales con su frontera, a excepción de la cubierta secundaria. Este procedimiento hace que  $I_S$  tenga un sentido horario de forma natural.

De acuerdo a la Figura 3.6, los nuevos polígonos quedan como:

$$O_S = \langle CH(A)_i = p_6, CH(A)_{i+1} = q_8, q_7, q_6, q_5 \rangle, \quad I_S = \langle q_8, p_6, p_7, p_8, p_9 \rangle .$$

Teniendo el polígono  $I_S$ , se obtiene su cubierta convexa y se guarda en la *deque*  $CH(I_S)$ . Al número de elementos de  $CH(I_S)$  se le denominará  $S$ , de igual forma que en el caso de  $CH(I)$ . No importa que se sobrescriba el valor anterior dado que ya no se utilizará más. Tal como en el caso de  $CH(I)$ , si se descarta el vértice que se repite al inicio y al final de  $CH(I_S)$ , así como los vértices de la cubierta secundaria inicial, se obtienen un número  $v$  de vértices, donde  $v = S - 3$ , los cuales se ingresan a  $CH(A)$  entre sus vértices  $CH(A)_i$  y  $CH(A)_{i+1}$ . Tal como se hizo anteriormente, la variable  $s$  que determina el número de vértices (o segmentos) de  $CH(A)$  debe actualizarse al haber agregado  $v$  nuevos vértices.

Hasta este momento, se han agregado a la *deque* inicial  $CH(A)$  un número  $w$  de vértices, donde  $w = u + v$ .

Seguidamente, de forma similar al proceso para formar los polígonos  $O_S$  e  $I_S$ , se forman los polígonos  $O_E$  e  $I_E$ . El primero se traza en sentido horario partiendo de la *cubierta secundaria final*, integrada por el segmento que une los vértices  $CH(A)_{i+w+1}$  y  $CH(A)_{i+w}$ , además de todos los vértices de  $B$  posteriores a  $CH(A)_{i+w}$  hasta el primer vértice fuera de la cavidad principal (o colineal con su cubierta).

El polígono  $I_E$  se forma en sentido horario iniciando con la cubierta secundaria final y agregando los vértices de  $A$  que estén dentro de  $O_E$  o que sean colineales con su frontera (exceptuando la cubierta secundaria). Seguidamente se obtiene  $CH(I_E)$ , que contiene  $S$  elementos. De ellos, solo se agregarán los  $x = S - 3$  vértices correspondientes entre los vértices  $CH(A)_{i+w}$  y  $CH(A)_{i+w+1}$  en  $CH(A)$ . La variable  $s$  debe actualizarse sumándole  $x$  unidades.

De acuerdo a la Figura 3.6, los nuevos polígonos quedan como

$$O_E = \langle CH(A)_{i+w+1} = p_{18}, CH(A)_{i+w} = q_{12}, q_{13}, q_{14}, q_{15} \rangle, \quad I_E = \langle p_{18}, q_{12}, p_{16}, p_{17} \rangle .$$

Definiendo  $y = u + v + x$ , se tiene que al final de este procedimiento se han añadido un número  $y$  de vértices a la lista del resultado  $CH(A)$ .

Con este proceso se agregan a la cubierta convexa relativa todos los vértices posibles para una cavidad en  $A$  con vértices de  $B$  dentro de ella. Sin embargo, después de este proceso pueden quedar cavidades en  $B$  con vértices de  $A$  dentro de ella. Cuando se presenta esta situación es necesario que por medio del avance de las iteraciones, el algoritmo llegue hasta esa cavidad e ingrese los vértices correspondientes al resultado. Para lograr esto, se realiza un proceso completamente análogo al descrito previamente, formando también los polígonos  $O$ ,  $I$ ,  $O_S$ ,  $I_S$ ,  $O_E$  e  $I_E$  y siguiendo la misma lógica.

En la Figura 3.8 se presenta un ejemplo de una cavidad en  $B$  con vértices de  $A$  dentro de ella.

En esta situación,  $O = \langle q_6, q_7, \dots, q_{17}, q_{18} \rangle$  e  $I = \langle q_{18}, q_6, p_8, p_9, \dots, p_{15}, p_{16} \rangle$ . La cubierta secundaria inicial está formada por  $\overline{q_6 p_{10}}$  y la cubierta secundaria final por  $\overline{q_{18} p_{14}}$ . Los demás polígonos necesarios pueden deducirse por analogía al proceso descrito anteriormente.

Cabe mencionar que después de que el algoritmo analiza una cavidad en  $A$  como la del ejemplo analizado, en el resultado pueden quedar aún algunas cavidades en  $A$ , resultantes de los vértices agregados mediante  $CH(I_S)$  y  $CH(I_E)$ . Por ejemplo, en el caso tratado se tiene una cavidad entre  $p_7$  y  $p_9$ , que son parte del resultado  $CH(A)$ . Aunque si bien en el ejemplo esta cavidad está vacía, si tuviera algún vértice de  $B$  dentro de ella, se aplicaría el mismo procedimiento descrito anteriormente cuando el proceso iterativo encuentre esta cavidad, es decir, cuando  $CH(A)_i = p_7$  y  $CH(A)_{i+1} = p_9$ .

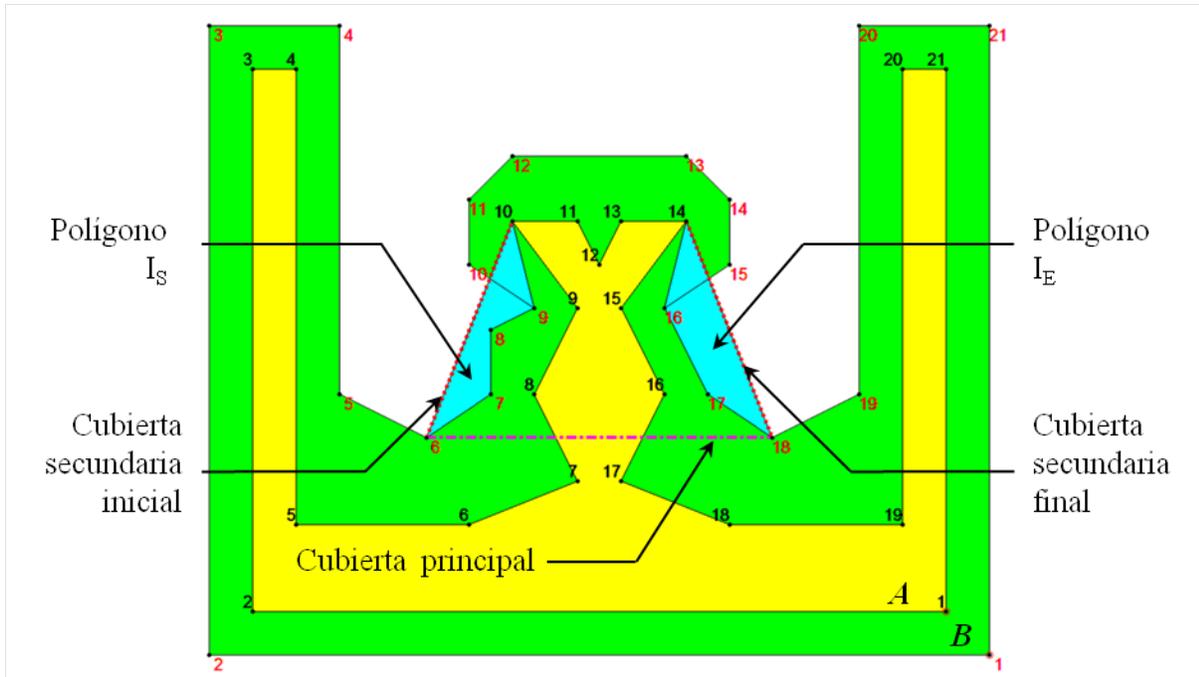


Figura 3.8: Cavidad en  $B$  con vértices de  $A$  dentro de ella.

Esto es posible porque después de realizar todo el proceso previamente descrito, el índice  $i$  permanece sin cambios, es decir, después de haber analizado una cavidad, se revisa entre todos los nuevos segmentos añadidos en busca de nuevas cavidades. Si inicialmente no se detecta una cavidad entre el par de vértices  $CH(A)_i$  y  $CH(A)_{i+1}$ , el índice  $i$  aumenta una unidad para verificar el siguiente par de vértices en  $CH(A)$ .

Una característica del algoritmo cuya justificación podría no ser evidente es la inclusión de vértices ubicados fuera de la cavidad principal al momento de generar los polígonos  $O_S$  y  $O_E$ . Recordando que el motivo de generar estos polígonos es la búsqueda de vértices de  $A$  que puedan pertenecer al resultado final, puede parecer extraño que al agregar estos vértices los polígonos  $O_S$  y  $O_E$  tengan una parte de su área fuera de la cavidad principal, donde es bien conocido que no pueden existir vértices que pertenezcan a la cubierta convexa relativa.

La razón de agregar estos vértices a  $O_S$  y  $O_E$  es formar una región que cubra completamente las zonas que el algoritmo de G. Klette no verifica.

Como se mencionó en la sección anterior, las zonas ciegas para el algoritmo de G. Klette están definidas por ciertos vértices de  $A$  y  $B$ , además de un vértice que resulta de la intersección de dos segmentos particulares ya descritos. Al agregar los vértices ubicados fuera de la *cubierta principal* se evita el procesamiento requerido para buscar los segmentos mencionados previamente y encontrar su punto de intersección. Además, de esta forma todos los cálculos se mantienen utilizando únicamente vértices de  $A$  y  $B$ .

El único costo que tiene el haber agregado estos vértices es que parte del área de los polígonos  $O_S$  y  $O_E$  se encuentre fuera de la cavidad principal. Sin embargo, dentro de estas regiones exteriores no existirán vértices que pertenezcan a la cubierta convexa relativa. Esto se debe a que el interior de esta región no puede contener vértices de  $A$  (que son los que se buscan al interior de  $O_S$  y  $O_E$ , y pueden ser parte de la cubierta convexa relativa) ya que si así fuera,

significaría que están fuera de la cubierta convexa de  $A$ . Esto es imposible porque si existieran vértices de  $A$  fuera de su cubierta convexa, estos deberían pertenecer a ella. En contraste, estas regiones exteriores sí pueden contener vértices de  $B$ , sin embargo, esto no afecta al resultado.

En la Figura 3.9 se ilustra la diferencia entre los polígonos  $O_{new}$  que forma el algoritmo de G. Klette a los extremos de cada cavidad, así como sus correspondientes zonas ciegas. También se muestran los polígonos  $O_S$  y  $O_E$  que genera la nueva versión del algoritmo, cada uno de los cuales se conforma de  $O_{new}$ , su zona ciega y una región que sobresale de  $CH(A)$ .

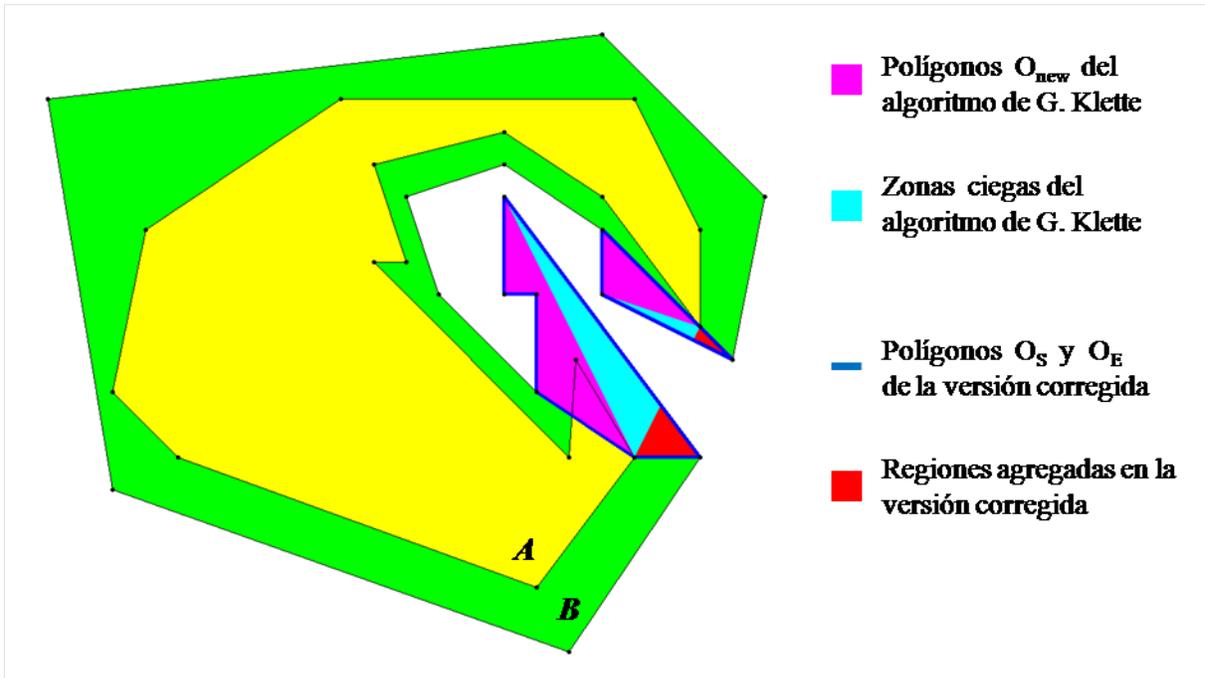


Figura 3.9: Diferencia entre los polígonos  $O_S$ ,  $O_E$  y  $O_{new}$ .

### 3.2.2. Pseudocódigo de la versión revisada

En esta sección se presenta el pseudocódigo de la nueva versión propuesta del algoritmo para determinar la cubierta convexa relativa de dos polígonos simples. En el apéndice A se muestra el diagrama de flujo de la versión implementada de esta propuesta.

---

**Entrada:** Dos polígonos simples  $A = \langle p_1, p_2, \dots, p_n \rangle$  y  $B = \langle q_1, q_2, \dots, q_m \rangle$ , tales que  $A \subseteq B$  (trazado en sentido horario).

**Salida:** Los vértices de la cubierta convexa relativa  $CH_B(A)$  en la lista  $CH(A)$  (trazado en sentido horario).

---

- 1: Inicializar  $CH(A) = \emptyset$ ,  $i = 1$ .
  - 2: Determinar la cubierta convexa de  $A$  mediante el algoritmo de Melkman, y guardarla en la *deque*  $CH(A)$ ;  $s$  es su número de elementos.
  - 3: Eliminar el primer elemento de  $CH(A)$ .
  - 4: Extender  $A$ ,  $B$  y  $CH(A)$  copiando su primer elemento al final.
  - 5: **while**  $i < s$  **do**
  - 6:     **if** Es detectada una cavidad entre  $CH(A)_i$  y  $CH(A)_{i+1}$ , **then**
  - 7:         **if** La cavidad está en  $A$ , **then**
  - 8:              $CAV(CH(A), i, s, A, B)$
  - 9:         **else**
  - 10:              $CAV(CH(A), i, s, B, A)$
  - 11:         **end if**
  - 12:     **end if**
  - 13:      $i=i+1$
  - 14: **end while**
  - 15: Se tiene la cubierta convexa de  $A$  relativa a  $B$  en la lista  $CH(A)$ .
-

---

```

1: procedure CAV( $CH(A)$ ,  $i$ ,  $s$ ,  $P1$ ,  $P2$ )
2:   Inicializar las variables locales  $u = 0, v = 0, w = 0$ .
3:   Generar el polígono  $O$  con todos los vértices de  $P1$ , desde  $CH(A)_i$  hasta  $CH(A)_{i+1}$ .
4:   Formar el polígono  $I$  con los vértices de la cubierta principal y todos los vértices de  $P2$ 
   dentro de  $O$  o colineales con su frontera (excepto vértices de  $P2$  colineales con la cubierta
   principal).  $N$  es su número de elementos.
5:   if  $N > 2$  then
6:     Determinar  $CH(I)$ .  $S$  es su número de elementos.
7:     Agregar entre  $CH(A)_i$  y  $CH(A)_{i+1}$  los vértices de  $CH(I)$ , exceptuando los que co-
   rresponden a la cubierta principal y al vértice que se repite.
8:      $u = S - 3$ 
9:      $s = s + u$ 
10:    Generar el polígono  $O_S$  con los vértices  $CH(A)_i$ ,  $CH(A)_{i+1}$  y todos los vértices de
    $P2$  previos a  $CH(A)_{i+1}$  hasta el primer vértice fuera de la cavidad principal.
11:    Formar el polígono  $I_S$  con la cubierta secundaria inicial y todos los vértices de  $P1$ 
   dentro de  $O_S$  o colineales con su frontera (excepto vértices de  $P1$  colineales con la cubierta
   secundaria inicial).  $N_S$  es su número de elementos.
12:    if  $N_S > 2$  then
13:      Determinar  $CH(I_S)$ .  $S$  es su número de elementos.
14:      Agregar entre  $CH(A)_i$  y  $CH(A)_{i+1}$  los vértices de  $CH(I_S)$  exceptuando los que
   corresponden a la cubierta secundaria inicial y al vértice que se repite.
15:       $v = S - 3$ 
16:       $s = s + v$ 
17:       $w = u + v$ 
18:    end if
19:    Generar el polígono  $O_E$  con los vértices  $CH(A)_{i+w}$ ,  $CH(A)_{i+w+1}$  y todos los vértices
   de  $P2$  posteriores a  $CH(A)_{i+w}$  hasta el primer vértice fuera de la cavidad principal.
20:    Formar el polígono  $I_E$  con la cubierta secundaria final y todos los vértices de  $P1$ 
   dentro de  $O_E$  o colineales con su frontera (excepto vértices de  $P1$  colineales con la cubierta
   secundaria final).  $N_E$  es su número de elementos.
21:    if  $N_E > 2$  then
22:      Determinar  $CH(I_E)$ .  $S$  es su número de elementos.
23:      Agregar entre  $CH(A)_{i+w}$  y  $CH(A)_{i+w+1}$  los vértices de  $CH(I_E)$  exceptuando los
   que corresponden a la cubierta secundaria final y al vértice que se repite.
24:       $x = S - 3$ 
25:       $s = s + x$ 
26:    end if
27:     $i = i - 1$ 
28:  end if
29:  return  $CH(A)$ ,  $i$ ,  $s$ 
30: end procedure

```

---

Figura 3.10: Pseudocódigo de la versión revisada.



# Capítulo 4

## Implementación y Resultados

La etapa de implementación es posiblemente una de las más complicadas al momento de verificar un algoritmo. Cuando se analiza solo un pseudocódigo, se tiene únicamente como limitante la claridad de este, su complejidad, así como el conocimiento sobre el tema y la capacidad de abstracción del analista. Sin embargo, cuando también se implementa el algoritmo surgen otras restricciones tales como las limitantes del propio lenguaje de programación, además de los conocimientos y la habilidad del programador. La gran ventaja de implementar un algoritmo es que este se puede poner a prueba con distintas variantes de datos entrada y de esta forma detectar más fácilmente posibles errores en el algoritmo.

En las siguientes secciones se describen las consideraciones que tuvieron que realizarse y los métodos utilizados para implementar tanto el algoritmo de Melkman para obtener la cubierta convexa, como la nueva versión propuesta para determinar la cubierta convexa relativa de dos polígonos simples.

### 4.1. Justificación del uso de Matlab

Para poder comprobar el correcto funcionamiento de la versión corregida con diversas configuraciones de polígonos y cavidades, esta fue implementado utilizando la plataforma de programación de Matlab. La versión utilizada para desarrollarlo fue Matlab R2012a, aunque el código ha sido ejecutado sin problemas desde versiones previas tales como la R2009b.

Las principales razones que motivaron el uso de este lenguaje se basan en que Matlab ofrece capacidades de programación de alto nivel, y facilidad para realizar operaciones matemáticas y de graficación dentro de un mismo entorno, a diferencia de otros lenguajes de programación tales como C/C++, Visual Basic, Java, entre otros, que generalmente están limitados en al menos uno de los aspectos mencionados antes, o que requieren de un gran dominio del lenguaje de programación para trabajar fluidamente. En particular:

- Matlab ofrece un manejo fluido de matrices y vectores, lo cual facilita mucho las operaciones con listas de coordenadas expresadas matricialmente.
- Matlab tiene capacidades de despliegue de gráficos potentes y flexibles.
- Matlab posee un lenguaje de alto nivel que incluye funciones para realizar operaciones matemáticas fácilmente.

- Matlab es un software común en el ámbito académico por su sencilla forma de programación y su gran capacidad de procesamiento de datos.
- Tomando en cuenta las restricciones de tiempo para desarrollar una tesis de maestría, se optó por utilizar y fortalecer los conocimientos en Matlab que el autor de este trabajo ya poseía, en lugar de invertir tiempo en aprender algún otro lenguaje en el que se tenían bases más débiles, y además no cumplían con todos los criterios descritos en los puntos anteriores.

## 4.2. Implementación del algoritmo de Melkman

En distintas etapas del algoritmo de G. Klette así como de la nueva versión, es necesario obtener la cubierta convexa de algún polígono. Aunque en ambos pseudocódigos solo se enuncia que debe obtenerse alguna cubierta convexa, no es una operación sencilla, al grado que es necesario ejecutar un algoritmo dedicado únicamente para esta tarea. Este algoritmo se ejecuta como una subrutina dentro del algoritmo principal.

Es importante aclarar en este punto que la versión del algoritmo de Melkman que fue implementada no corresponde de forma fiel ni al pseudocódigo de la fuente original [8], ni a la versión presentada en [7]. Esto se debe a diversos motivos, los cuales se enlistan a continuación:

En [8], el pseudocódigo:

- utiliza la operación DO/UNTIL, la cual indica un proceso iterativo. Sin embargo, el uso de esta instrucción es poco común, incluso en la literatura. Esto puede dificultar la comprensión del algoritmo.
- detecta una concavidad o convexidad local asignándole un valor (-1, 0 o 1) a una tripleta de vértices según el tipo de vuelta que forman. Esta designación se realiza de acuerdo a una relación que solo se describe verbalmente.
- opera para un polígono definido en sentido antihorario.
- obtiene el resultado correcto aun cuando la primera tripleta de vértices del polígono de entrada forma una *vuelta derecha*. Esto equivale a una *vuelta izquierda* cuando se considera un sentido horario (tal como se hace en esta tesis).

En contraste, en [7] el pseudocódigo:

- utiliza la operación iterativa WHILE, la cual es bastante común y fácil de comprender.
- detecta una concavidad o convexidad local asignándole un valor (positivo, negativo o cero) a una tripleta de vértices según el tipo de vuelta que forman. Esta designación se realiza de acuerdo al resultado de una operación de determinante.
- opera para un polígono definido en sentido horario.

- puede obtener un resultado incorrecto cuando la primera tripleta de vértices del polígono de entrada forma una *vuelta derecha*, puesto que se omiten algunas instrucciones iniciales del pseudocódigo original.

De acuerdo a las diferencias descritas previamente, el algoritmo implementado está basado mayormente, por su claridad y sentido de operación, en el pseudocódigo de [7], pero incluyendo las líneas faltantes que sí están presentes en [8] para asegurar su correcto funcionamiento. En la figura 4.1 se observa el resultado incorrecto producto de la omisión de algunas líneas de la fuente original en el algoritmo de [7].

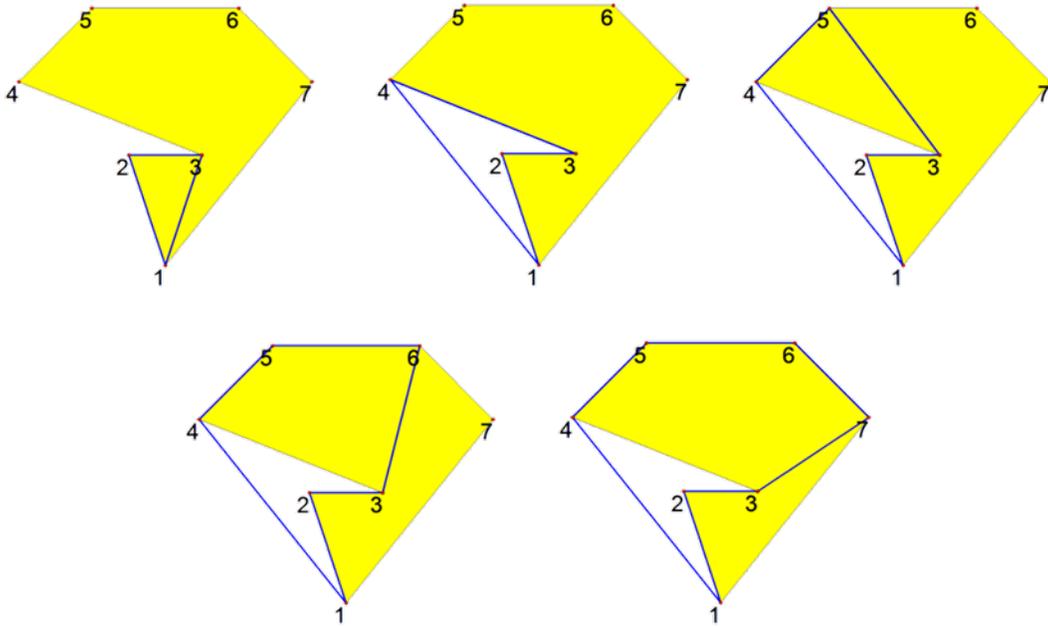


Figura 4.1: Error producido por las instrucciones faltantes en el pseudocódigo de [7].

En la figura 4.2 se muestra el pseudocódigo utilizado en esta tesis para implementar el algoritmo de Melkman.

Como se mencionó anteriormente, el lenguaje de programación también impone restricciones al momento de implementar un algoritmo. A continuación se describen algunas adaptaciones que fueron necesarias para adecuar algunas instrucciones del pseudocódigo anterior al lenguaje de Matlab.

La adaptación más importante se hizo al cambiar las polilíneas y las *deques* a una forma matricial. Es importante recordar que el algoritmo de Melkman tiene como dato de entrada una polilínea, no un polígono. En la descripción del algoritmo en [8], una polilínea queda representada por una lista ordenada de vértices, e.g.  $P = \langle v_1, v_2, \dots, v_m \rangle$ . El dato de salida del algoritmo es un polígono, representado mediante una *deque*, e.g.  $D = \langle d_{bottom}, d_{bottom+1}, \dots, d_{top-1}, d_{top} \rangle$  con  $d_{bottom} = d_{top}$ .

Para dar mayor sentido a los términos *top* y *bottom* de las *deques*, se decidió utilizar vectores columna para representar la polilínea de entrada y la *deque* de salida. De acuerdo a las descripciones de [8] y [7], los índices se incrementan desde *bottom* hasta *top*, tal como si se tratara

---

**Entrada:** Una polilínea simple  $\rho = (p_0, p_1, \dots, p_{n-1})$  con  $n$  vértices en el plano.  
**Salida:** La lista ordenada de los vértices de la cubierta convexa de la polilínea de entrada.

---

```
1: if  $D(p_0, p_1, p_2) < 0$  then
2:   PUSH  $p_0$  y luego  $p_1$  en la deque. PUSH  $p_2$  e INSERT  $p_2$ .
3: else
4:   PUSH  $p_1$  y luego  $p_0$  en la deque. PUSH  $p_2$  e INSERT  $p_2$ .
5: end if
6: Sea  $k = 3$ 
7: while  $k < n$  do
8:   while  $D(q_{bot}, q_{bot+1}, p_k) > 0$  AND  $D(q_{top-1}, q_{top}, p_k) > 0$  AND  $k < n$  do
9:      $k = k + 1$ 
10:  end while
11:  while  $D(q_{bot}, q_{bot+1}, p_k) \leq 0$  do
12:    DELETE
13:  end while
14:  INSERT  $p_k$ 
15:  while  $D(q_{top-1}, q_{top}, p_k) \leq 0$  do
16:    POP
17:  end while
18:  PUSH  $p_k$ ,  $k = k + 1$ 
19: end while
20: Los vértices de la cubierta convexa se encuentran en la deque.
```

---

Figura 4.2: Pseudocódigo del algoritmo de Melkman utilizado en la implementación.

de la numeración de los pisos de un edificio. Sin embargo, en la implementación la numeración se hizo de forma inversa, es decir, comenzando con *top* e incrementando hasta *bottom*, ya que es más natural visualizar una lista que se incrementa hacia abajo. De esta forma, el índice del vértice *top* siempre es igual a 1, y el de *bottom* es igual a la cardinalidad de la *deque*  $D$ , la cual aquí se denominará  $b$ .

Una vez que se ha convenido la orientación y el sentido de la *deque*, es necesario desglosar cada elemento. Cada vértice  $v_i$  está integrado por tres componentes: su índice  $i$ , su coordenada  $x$ , y su coordenada  $y$ . Con cada uno de estos componentes se forma una columna.

A pesar de que estas tres columnas son suficientes para representar de forma completa una polilínea, para facilitar algunas operaciones del algoritmo es necesario añadir una columna de números uno a la derecha de las columnas que definen las coordenadas de los vértices. El motivo de incluir esta columna es facilitar la obtención del determinante que corresponde a la llamada *prueba de Sklansky* descrita en la Sección 1.5.3, utilizada para conocer el tipo de vuelta de una tripleta de vértices.

En la figura 4.3 se muestra un polígono simple (izquierda) y su representación en la forma matricial que se ha descrito (derecha). Para obtener el determinante antes mencionado basta con definir la submatriz que se marca en el recuadro de la figura, y después solo aplicarle un comando predefinido de Matlab para obtener su determinante. Esta operación sería más compleja en cualquier otro lenguaje de programación común.

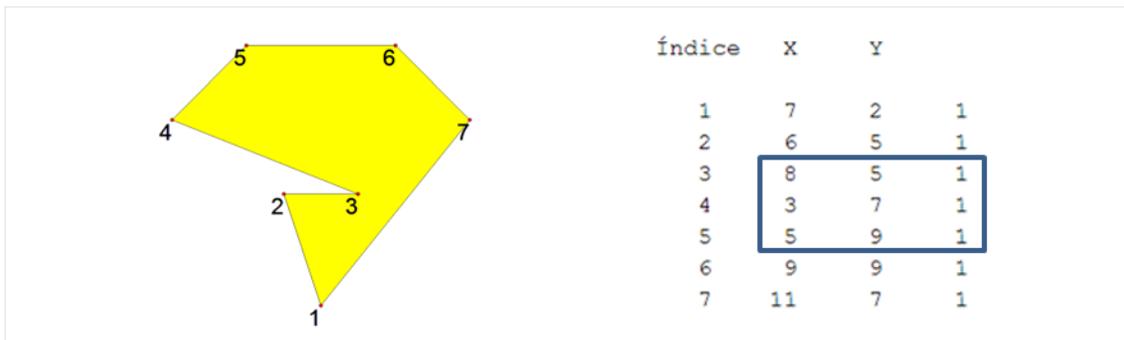


Figura 4.3: Representación de un polígono en forma matricial.

Como se mencionó antes, el lenguaje de programación también impone restricciones al momento de implementar un algoritmo, y este caso no es la excepción. A pesar de que se combinó el pseudocódigo de [8] y el de [7] para formar uno sencillo, comprensible, correcto y fácil de programar, surgieron algunos detalles que impiden que su implementación sea directa.

El problema principal surge cuando en la línea 9 del pseudocódigo de la figura 4.2 se incrementa  $k$  de tal forma que  $k = n$ . Si esto sucede, en el siguiente ciclo el programa buscará el (inexistente) vértice  $p_n$ , el cual es necesario para verificar la condición del ciclo. Al no existir este vértice se produce un error y el programa se detiene. Incluso si se corrigiera esta situación para este ciclo iterativo, se produciría el mismo error para los ciclos posteriores que también necesitarían operar con el vértice  $p_n$ . La solución a esto fue que cuando ocurra esta situación en la línea 9, el programa salga de ambos ciclos anidados y despliegue el resultado final.

Otro problema relevante apareció al momento de hacer pruebas con polígonos que contenían vértices colineales. Al realizar pruebas con el código se observó que estas tripletas colineales eran

interpretadas en algunas ocasiones como si formaran una vuelta derecha, y en otras como una vuelta izquierda. Después de un análisis detallado se llegó a la conclusión de que el problema radicaba en el propio Matlab. Tal deducción fue evidente después de revisar los resultados de los determinantes de las tripletas de vértices colineales. Es necesario recordar en este punto que cuando una triplete de vértices es colineal, su determinante es cero. Durante la revisión se observaron resultados con magnitudes que si bien eran despreciables (ordenes menores a  $10^{-10}$ ), no eran cero. Estas pequeñas variaciones afectaban directamente al resultado del algoritmo.

Este tipo de problemas hace evidente que a pesar de la gran capacidad de procesamiento y los avanzados algoritmos utilizados por Matlab, este software sigue estando limitado por los problemas computacionales típicos producidos por los métodos numéricos, incluso para operaciones tan básicas como un determinante de números enteros.

Otros problemas menores surgieron a partir del cambio de perspectiva en los índices de la polilínea de entrada y de la *deque* de salida, pero fueron resueltos fácilmente.

### 4.3. Implementación de la versión revisada

La implementación del algoritmo que se propone en este trabajo no tuvo inconvenientes importantes por dos razones principales. La primera es que una parte importante del algoritmo está basado en el de Melkman, el cual ya había sido implementado satisfactoriamente. La única modificación en este aspecto fue la inclusión de una nueva columna en la representación matricial de los polígonos. Esta nueva columna es de números uno si es la matriz del polígono interior, y de números dos si es la matriz del polígono exterior. Estas columnas permiten identificar a qué polígono corresponde un vértice dentro del algoritmo y en el resultado final.

La segunda razón tiene que ver con el desarrollo mismo del algoritmo. Como se dijo antes, esta propuesta surgió después de modificar y corregir progresivamente lo que pretendía ser una implementación del algoritmo de G. Klette. Así, la versión revisada que se propone surgió a partir del código ya desarrollado, y no de una nueva metodología u otro pseudocódigo base.

Una vez que se tuvo un código funcional, se bosquejó el pseudocódigo. Sin embargo, era bastante más largo que el del algoritmo de G. Klette. Aprovechando que algunas de las operaciones eran similares o análogas entre sí, fue posible simplificarlo utilizando las mismas subrutinas en diversas etapas del algoritmo. Al hacer esto fue posible también modificar y reducir el código. Estas operaciones se repitieron en algunas ocasiones hasta obtener el pseudocódigo final presentado en la Sección 3.2.2.

### 4.4. Interfaz gráfica

Un aspecto importante de la implementación de ambos algoritmos descritos previamente, es la inclusión de rutinas que generan una interfaz gráfica que facilita la inserción de datos de entrada y la graficación de los resultados. Esta interfaz también es de gran ayuda durante la etapa de desarrollo, dado que permite visualizar resultados intermedios a lo largo de distintas etapas del algoritmo, y así detectar posibles errores en el código.

Si bien estas rutinas no son propiamente parte de los algoritmos, ni afectan su estructura en forma alguna, son indispensables para poder visualizar el correcto funcionamiento de los mismos.

Para el algoritmo propuesto no solo se desarrolló una interfaz con el fin de visualizar resultados, sino también con el objetivo de proporcionar la siguiente información al usuario:

- Descripción básica y objetivos del algoritmo.
- Consideraciones importantes del algoritmo y restricciones del programa.
- Opciones sobre el método de ingreso de los datos de entrada.
- Instrucciones sobre el formato correcto de los datos de entrada.
- Despliegue de la lista de vértices del resultado final ( $CH(A)$ ), al término de la ejecución del programa.

Toda esta información está organizada en forma de una carátula inicial y un menú de opciones.

A pesar de que Matlab ofrece un ambiente de desarrollo de aplicaciones de manera más gráfica (*GUI Builder*), este menú inicial aparece en texto con formato sencillo dentro del mismo *workspace* de Matlab. La razón de esto es que el menú desempeña una función meramente informativa y de orientación para el uso correcto del programa. El despliegue de resultados no es afectado al utilizar este tipo de menú. Aunque al haber utilizado el *GUI Builder* se habría logrado una interfaz más integrada y con mejor presentación, el código se haría mucho más extenso y complejo debido a la gran cantidad de estructuras necesarias para el manejo de textos, imágenes, botones, ventanas, etc.

## 4.5. Resultados de la versión revisada

En esta sección se ilustran y describen a detalle los casos en los que la versión revisada funciona correctamente, comparándolos con los resultados que se esperaría obtener aplicando el algoritmo de G. Klette. En la figura 4.4 se presenta el ejemplo a partir del cual se realizará la comparativa de resultados.

Esta figura fue diseñada de tal forma que incluyera varios de los casos típicos, y también algunos casos que podrían resultar difíciles de resolver satisfactoriamente para cualquier algoritmo que pretenda determinar la cubierta convexa relativa de dos polígonos simples en el plano.

Como puede observarse, la figura es básicamente rectangular. En su parte superior ( $p_1 - p_{21}$ ) se pueden identificar los siguientes casos:

- Una cavidad convexa en  $A$  ( $p_1 - p_4$ ) con puntos de  $B$  dentro de ella.
- Una cavidad no convexa en  $A$  ( $p_4 - p_{14}$ ) con puntos de  $B$  dentro de ella.
- Una cavidad no convexa en  $A$  ( $p_{14} - p_{21}$ ) con puntos de  $B$  dentro de ella, y una cavidad no convexa en  $B$  ( $q_{11} - q_{16}$ ) con puntos de  $A$  dentro de ella. Se presenta un anidamiento doble.

En su parte lateral derecha ( $p_{21} - p_{33}$ ) se encuentran los siguientes casos:

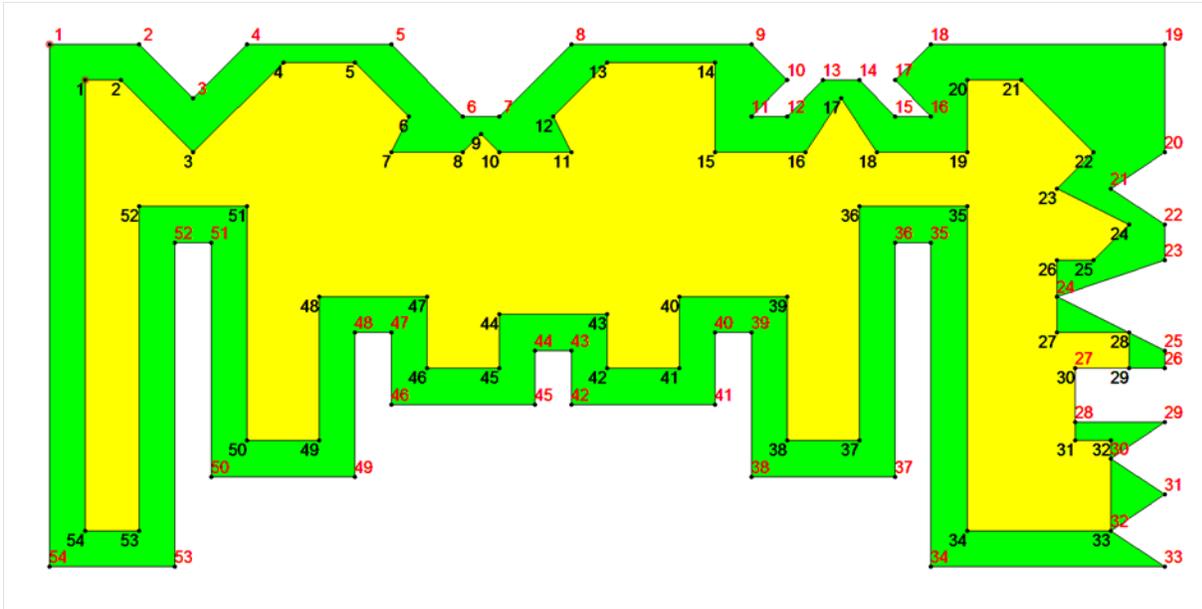


Figura 4.4: Polígonos de ejemplo para prueba comparativa.

- Una cavidad en  $A$  ( $p_{22} - p_{24}$ ), con un vértice de  $B$  colineal con la *cubierta principal* de la cavidad  $CAV(A)$ .
- Una cavidad en  $A$  ( $p_{24} - p_{29}$ ), con un vértice de  $B$  colineal con un lado de la cavidad  $CAV(A)$ . También, se tiene un vértice de  $A$  ( $p_{25}$ ) colineal con la *cubierta secundaria inicial*.
- Una cavidad en  $A$  ( $p_{29} - p_{33}$ ), en la que  $B$  es colineal con la frontera de la cavidad  $CAV(A)$  en un lado completo, uno parcial y dos vértices.
- Dos cavidades de  $B$  dentro de una sola cavidad  $CAV(A)$ , incluyendo distintos tipos de colinealidades.

Finalmente, en la parte inferior de los polígonos de prueba ( $p_{33} - p_{54}$ ) se tiene el caso de cavidades anidadas varias veces, con lados isotéticos.

En la figura 4.5 se puede observar que el resultado obtenido aplicando la nueva versión del algoritmo es el correcto, y puede corroborarse revisando la lista de vértices del resultado final que despliega el algoritmo.

En la figura 4.6 se muestra el resultado que se esperaría tener aplicando el algoritmo de G. Klette de acuerdo a la metodología descrita en [2]. Es importante recalcar que este algoritmo no se implementó, y la figura mostrada es solo un resultado *estimado*, además de que para este caso particular con fines comparativos, se supone que la metodología del algoritmo de G. Klette se aplica a los polígonos definidos en sentido horario. Como es evidente, el resultado obtenido es incorrecto.

A continuación se presenta una nueva prueba comparativa donde se involucran polígonos isotéticos. De acuerdo a [2] el algoritmo de G. Klette debería funcionar correctamente en estos casos.

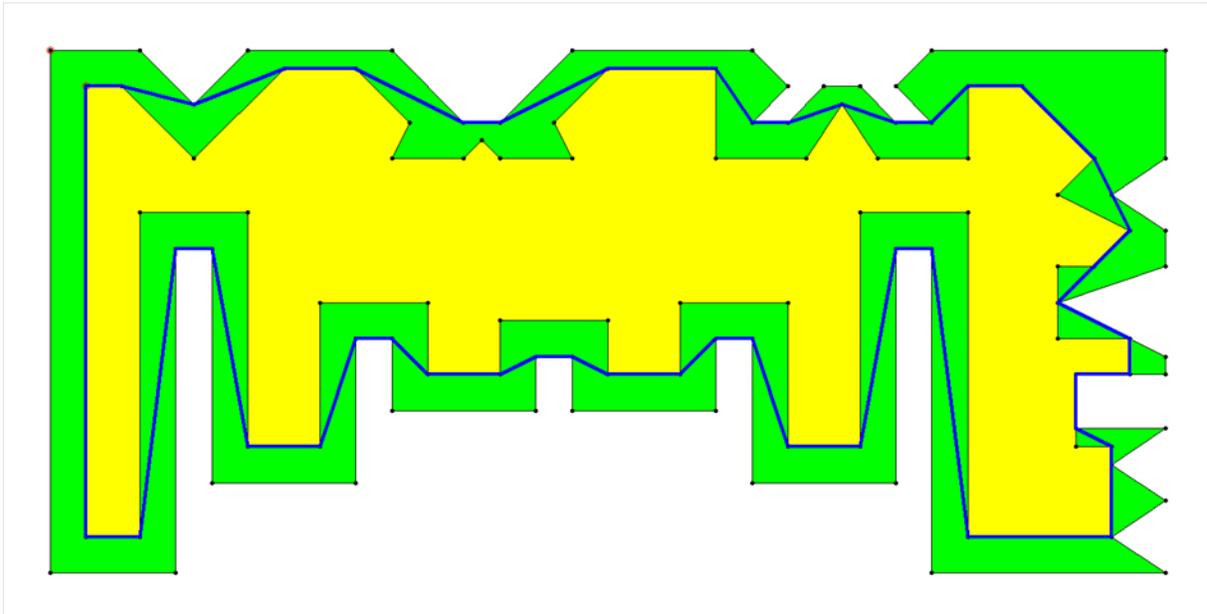


Figura 4.5: Resultado obtenido aplicando la versión corregida.

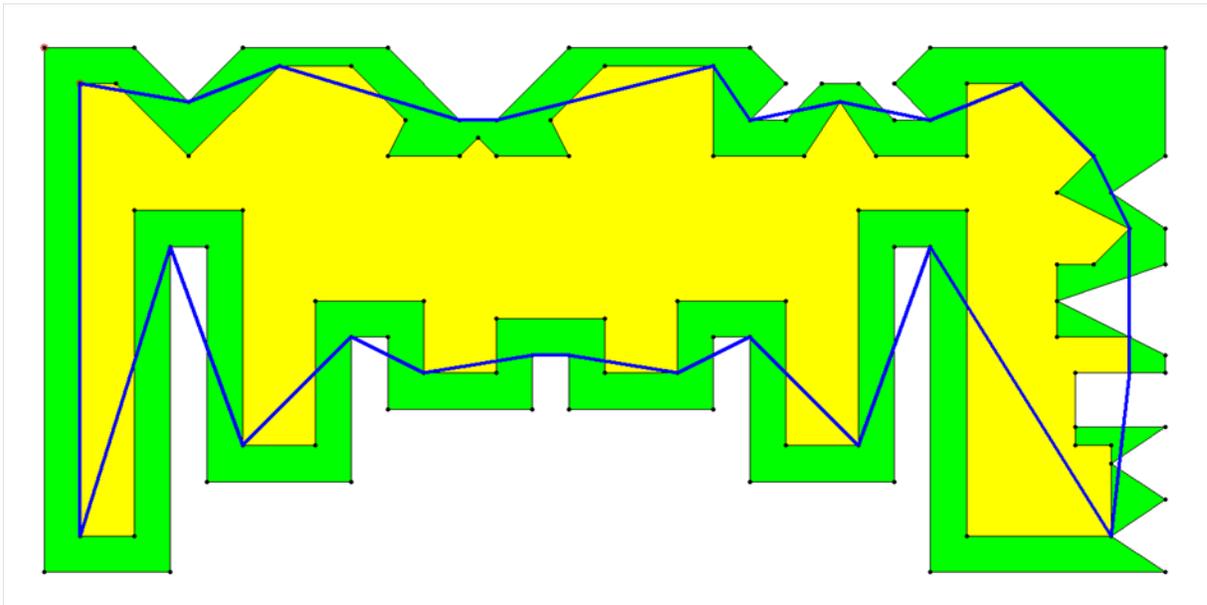


Figura 4.6: Resultado estimado aplicando el algoritmo de G. Klette.

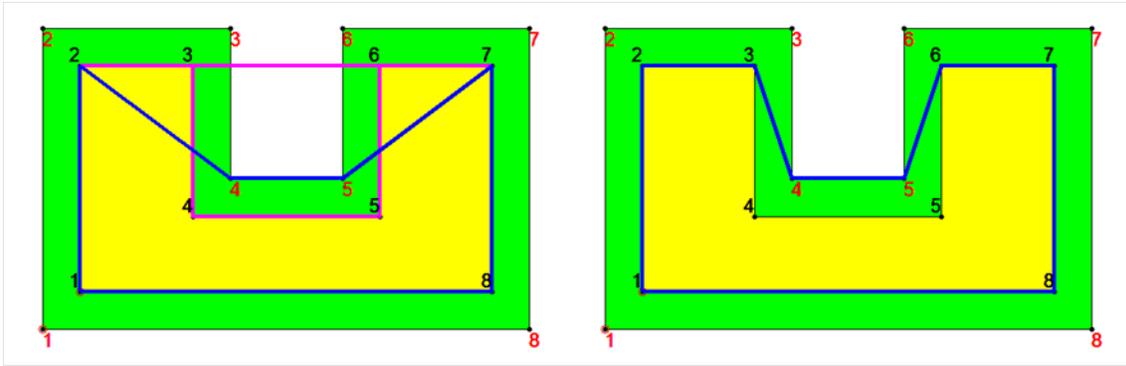


Figura 4.7: Resultados aplicando el algoritmo de G. Klette (izquierda) y la versión corregida (derecha) a polígonos isotéticos.

Como puede observarse en la figura 4.7, y tal como se había comentado en la Sección 3.1.2, el algoritmo de G. Klette falla también en estos casos. Cabe destacar que el polígono  $O$  formado entre  $p_2$  y  $p_7$  ya no es un polígono simple, e inicialmente se establece que el algoritmo no contempla la solución correcta en estos casos. Sin embargo, esta situación puede presentarse incluso en procesos intermedios cuando se trata con polígonos no isotéticos.

## 4.6. Discusión de resultados

Al observar las figuras 4.5, 4.6 y 4.7, los siguientes puntos se hacen evidentes:

- Los resultados obtenidos utilizando la nueva versión del algoritmo son correctos para todos los casos típicos y atípicos analizados. En contraste, los resultados obtenidos con el algoritmo de G. Klette no es correcto en todos los casos.
- El algoritmo de G. Klette falla incluso en algunas cavidades relativamente sencillas por causa de las *zonas ciegas* descritas en la Sección 3.1.2.
- El algoritmo de G. Klette falla al no detectar vértices que pertenecen a la cubierta convexa relativa cuando estos son colineales con la frontera de una cavidad.
- Aunque en [2] nunca se menciona que el algoritmo funcione de forma correcta en casos que involucran colinealidades, esto debería estar implícito puesto que sí se postula que es un algoritmo para el caso general de dos polígonos simples en el plano, tales que  $A \subseteq B$ .
- El algoritmo de G. Klette falla para el caso de polígonos isotéticos de la forma que se muestra en la figura 4.7. En esta situación es importante el hecho de que en el transcurso del algoritmo se forman polígonos no simples, siendo que el algoritmo especifica estar diseñado solo para el caso de polígonos simples. Sin embargo, a pesar de esta *excusa conceptual*, la verdadera falla radica en las ya mencionadas *zonas ciegas* de este algoritmo. Como puede verse en la misma figura 4.7, la nueva versión propuesta funciona correctamente también en estos casos.

- Con respecto a las cavidades anidadas, la metodología de ambos algoritmos conlleva a un resultado correcto. Sin embargo, es muy probable que el algoritmo de G. Klette obtenga resultados erróneos en este tipo de configuraciones debido a las fallas producidas por las zonas ciegas.



# Conclusiones

A partir del análisis realizado al algoritmo de G. Klette, en el cual se confirma que obtiene resultados erróneos en diversas situaciones, es posible concluir lo siguiente:

- Este algoritmo puede no obtener el resultado correcto por dos razones principales:
  - Durante cada paso de recursión, se generan zonas ciegas, las cuales pueden incluir vértices que pertenecen al resultado final.
  - No contempla el caso en el que se presentan colinealidades entre vértices de ambos polígonos (interior y exterior) dentro de una misma cavidad. Frecuentemente, estos vértices pertenecen al resultado final.
- Además de detectar los errores antes descritos, se realizaron diversas observaciones al pseudocódigo, así como a algunas afirmaciones realizadas en [2]. Las fallas mencionadas, así como las observaciones hechas al algoritmo se describen a detalle en la Sección 3.1.
- La implementación se dificultó por el uso de argumentos en la función recursiva, y por la necesidad de generar nuevas variables para un número desconocido de pasos recursivos.
- Se intentó modificar el algoritmo para corregir las fallas y observaciones antes descritas. Esto conllevó a la propuesta de una versión corregida del algoritmo, la cual posee una estructura iterativa, pero conserva la metodología descrita en [2].

Sobre la nueva versión propuesta se puede decir que:

- Corrige el problema de zonas ciegas y colinealidades del algoritmo de G. Klette. Además, por su estructura iterativa, no presenta los inconvenientes resultantes del proceso recursivo.
- En cuanto a resultados, la versión revisada supera al de G. Klette, de acuerdo a la discusión de la Sección 4.6.

En resumen, todos los objetivos planteados inicialmente fueron cumplidos. Algunos incluso fueron superados al proponer e implementar una nueva versión del algoritmo analizado que obtiene resultados satisfactorios para diversas clases de configuraciones comunes, tales como las presentadas en el Capítulo 4.



# Trabajos futuros

Como trabajos que pueden complementar o ampliar el alcance de esta tesis se pueden mencionar, sin ser limitativos, los siguientes:

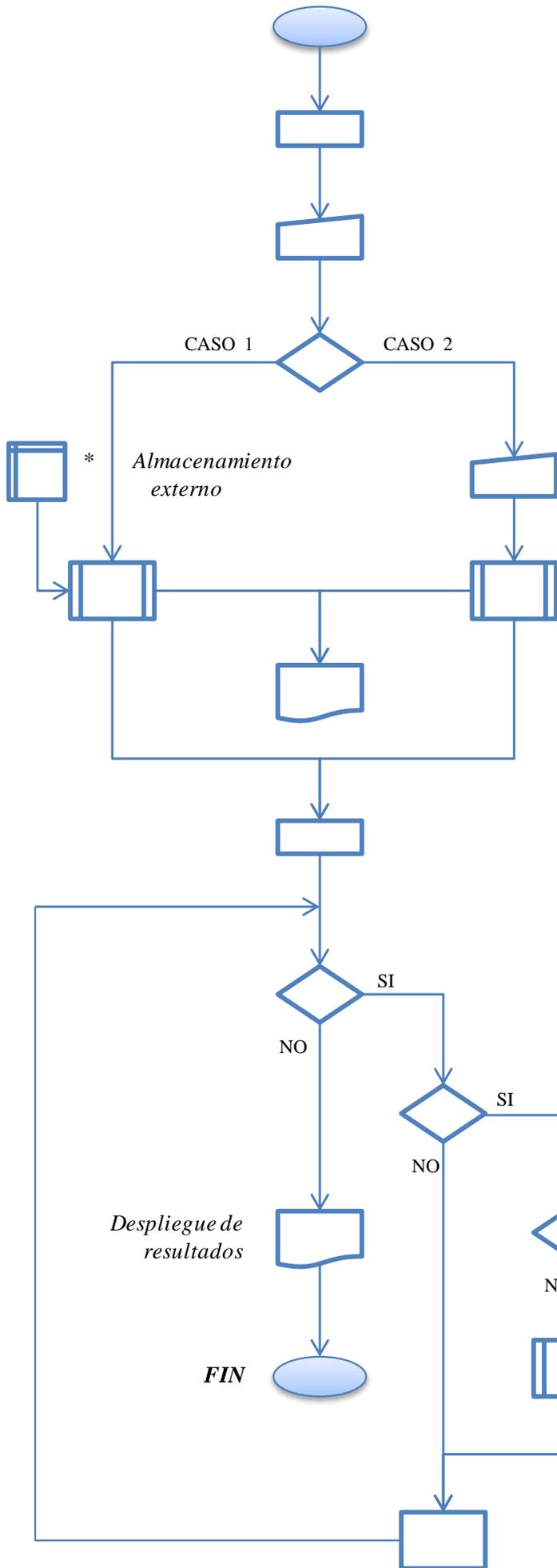
- Realizar un análisis detallado de funcionamiento para demostrar que la nueva versión es correcta, o para demostrar las condiciones bajo las cuales es correcta.
- Implementar la versión corregida en una plataforma de programación abierta o más universal para ponerla a disposición de la comunidad estudiantil y de investigación.
- Buscar más posibles aplicaciones de la cubierta convexa relativa dentro de áreas de control, robótica y/o análisis de imágenes, las cuales son áreas de interés dentro del Departamento de Control Automático.
- Estudiar y analizar el problema de la cubierta convexa relativa para poliedros en el espacio, y verificar si la nueva versión propuesta es escalable o si su metodología puede aprovecharse en algún sentido para resolver este problema.
- Proponer una metodología (o algoritmo) para determinar la cubierta convexa relativa ya no de polígonos simples, sino de curvas arbitrarias cerradas que puedan describirse mediante una cantidad finita de puntos de control (e.g. curvas de Bézier, *b-splines*).



# Apéndice A

## DIAGRAMA DE FLUJO DEL NUEVO ALGORITMO





**INICIO**

*Despliegue de información y menú de opciones*

*Elección del formato de datos de entrada*

*¿Datos de entrada mediante vectores de coordenadas previamente definidos (Caso 1) o introducción manual mediante el mouse (Caso 2)?*

*Adquisición de los datos de entrada (polígonos A y B).*

*Determinación de CH(A) (Algoritmo de Melkman)*

*Despliegue gráfico de polígonos A, B y CH(A).*

*Preprocesamiento de A, B y CH(A), e inicialización de variables.*

*¿  $i < s$  ?*

*¿ Cavidad detectada entre  $CH(A)_i$  y  $CH(A)_{i+1}$  ?*

*¿ La cavidad está en A?*

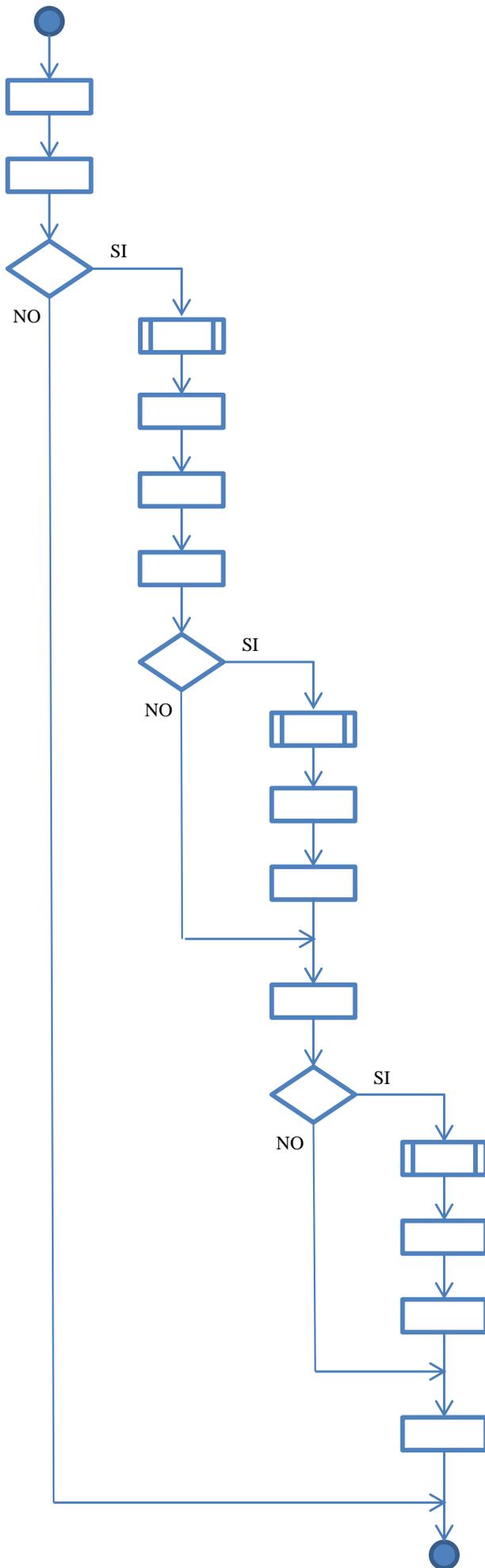
**Subrutina**

1.-  $CAV( CH(A), i, s, A, B )$

2.-  $CAV( CH(A), i, s, B, A )$

\* (ver desglose en siguiente página)

$i = i + 1$



*Inicialización de variables.*

*Generación del polígono O, y del polígono I.*

*¿ N > 2 ?*

*Determinación de CH(I).*

*Agregar CH(I) entre CH(A)<sub>i</sub> y CH(A)<sub>i+1</sub>.*

*Actualización de variables.*

*Generación del polígono O<sub>S</sub>, y del polígono I<sub>S</sub>.*

*¿ N<sub>S</sub> > 2 ?*

*Determinación de CH(I<sub>S</sub>).*

*Agregar CH(I<sub>S</sub>) entre CH(A)<sub>i</sub> y CH(A)<sub>i+1</sub>.*

*Actualización de variables.*

*Generación del polígono O<sub>E</sub>, y del polígono I<sub>E</sub>.*

*¿ N<sub>E</sub> > 2 ?*

*Determinación de CH(I<sub>E</sub>).*

*Agregar CH(I<sub>E</sub>) entre CH(A)<sub>i+w</sub> y CH(A)<sub>i+w+1</sub>.*

*Actualización de variables.*

*i = i - 1*

# Apéndice B

ARTÍCULO DE G. KLETTE



# Recursive Calculation of Relative Convex Hulls

Gisela Klette

Auckland University of Technology,  
School of Computing & Mathematical Sciences,  
Private Bag 92006, Auckland 1142, NZ

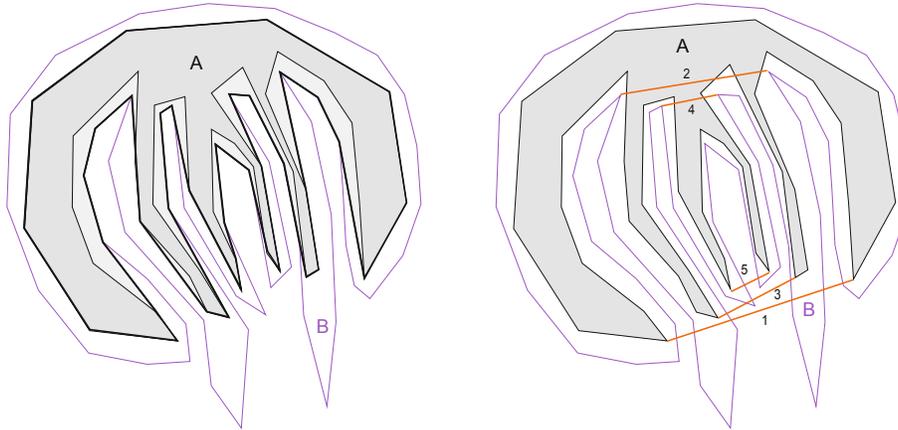
**Abstract.** The relative convex hull of a simple polygon  $A$ , contained in a second simple polygon  $B$ , is known to be the minimum perimeter polygon (MPP). Digital geometry studies a special case:  $A$  is the inner and  $B$  the outer polygon of a component in an image, and the MPP is called minimum length polygon (MLP). The MPP or MLP, or the relative convex hull, are uniquely defined. The paper recalls properties and algorithms related to the relative convex hull, and proposes a (recursive) algorithm for calculating the relative convex hull. The input may be simple polygons  $A$  and  $B$  in general, or inner and outer polygonal shapes in 2D digital imaging. The new algorithm is easy to understand, and is explained here for the general case. Let  $N$  be the number of vertices of  $A$  and  $B$ ; the worst case time complexity is  $\mathcal{O}(N^2)$ , but it runs for “typical” (as in image analysis) inputs in linear time.

**Keywords:** relative convex hull, minimum perimeter polygon, minimum length polygon, shortest path, path planning.

## 1 History and Outline of the Paper

Studies about relative convex hulls started in the 1970s in image analysis, robotics and computer vision. Sklansky [6] proposed the notion of the relative convex hull at a time when digital imaging was still in its infancy, and digital images of low resolution. The relative convex hull is known to be identical to the minimum perimeter polygon (MPP). This polygon circumscribes a given set within an available (polygonal) domain. Figure 1 shows on the left an inner polygon  $A$ , an outer polygon  $B$ , and the convex hull of  $A$  relatively to  $B$ ; the image on the right in this figure illustrates stacked cavities for those two polygons.

The relative convex hull generalizes the concept of the convex hull. Algorithms for the computation of convex hulls of simple polygons are basic procedures in computational geometry or digital image analysis. For example, Melkman [4] proposed a linear time algorithm for connected simple polylines; a *simple polyline* consists of subsequent (connected) line segments, without any crossing. The end vertex of one line segment coincides with the start vertex of the following line segment, possibly apart from the start and end vertex of the simple polyline (i.e., the simple polyline does not need to form a loop). A *simple polygon* is defined by a simple polyline if this forms a loop. The polyline is then the *frontier* of this polygon. The Melkman algorithm works efficiently for any simple polyline by



**Fig. 1.** An example of polygons  $A$  and  $B$  having stacked cavities. Left: Convex hull of  $A$  relatively to  $B$  (i.e., the *relative convex hull*) shown by the bold line. Right: Covers of stacked cavities.

using a *deque* (i.e., a double-ended queue) while computing the convex hull. We use this efficient algorithm as part of our new algorithm for the computation of the *relative convex hull* of a simple polygon. This convex hull is defined relatively to a second (larger) simple polygon.

Planning a shortest path for a robot in a restricted two-dimensional (2D) environment may also be solved by computing the relative convex hull of one simple inner polygon with respect to an outer simple polygon: The outer polygon might be defined by corners (vertices) of the available area, and the inner polygon by corners (vertices) of the obstacles. Obviously, inner and outer polygons in this case differ from polygons in image analysis. Here, vertices are in the real plane, edges are not limited to be isothetic, and the “width” of the space between inner and outer polygon is not constrained a-priori. Some publications about relative convex hulls are discussing geometric properties, but not algorithms, such as [8] or [9]. Proposed algorithms will be briefly reviewed below, before discussing a new algorithm. The paper is restricted to simple polygons. (The inner polygon could also be replaced by sets of points or other data.)

The new algorithm is for the general case of simple polygons (such as in the robotics scenario), but also for more constrained polygons such as in the digital imaging example. The relative convex hull provides a way of multigrid-convergent length measurement in digital imaging [2], thus making full use of today’s high resolution image data. In this particular context, the MPP is called *minimum length polygon* (MLP), and defines a multigrid-convergent method for length estimation. This method is an alternative way to purely local counts (e.g., counting edges along a digital frontier of an object). Local counts do not support more accurate length estimates in general when increasing the grid resolution. The advantage of having high resolution camera technology available would be wasted in image analysis if a method is applied that is not multigrid convergent. For showing multigrid convergence of the MLP towards the true perimeter, assume that a measurable set in the Euclidean plane is digitized

(Jordan digitization; see [2]) into an inner and an outer polygon. The relative convex hull (or MLP) is then calculated such that it contains the inner polygon, but itself is contained in the outer polygon. It has been shown [3] that the perimeter of this relative convex hull is converging towards the perimeter of the digitized measurable set with increasing the grid resolution.

First, the paper briefly recalls basic definitions and properties that are preliminaries to explain the algorithmic methods. We review existing algorithms for the computation of the relative convex hull. Next we list and show new theoretical results. They allow us to propose a completely new algorithmic approach for calculating the relative convex hull (or the MLP).

## 2 Basics

A simple polygon is bounded by a circular chain of co-planar line segments such that two segments only intersect at end (or start) vertices, and only if they are consecutive segments in the given circular chain. This circular chain is also called the *frontier* of the simple polygon.

A subset  $S \subset R^2$  of points is convex iff  $S$  is equal to the intersection of all half planes containing  $S$ . The *convex hull*  $\text{CH}(S)$  of a set of points  $S$  is the smallest (by area) convex polygon  $P$  that contains  $S$ .

### 2.1 The Relative Convex Hull

A finite set of  $n$  points in the plane can be associated with the set of  $n$  vertices of a simple polygon. The convex hull of a simple polygon  $A$  is a simple polygon  $\text{CH}(A)$ , and this has a reduced number of vertices if  $A$  is not convex. A simple polygon is also called a *Jordan polygon* because its frontier is a Jordan curve.<sup>1</sup>

**Definition 1.** A cavity of a polygon  $A$  is the topological closure of any connected component of  $\text{CH}(A) \setminus A$ .

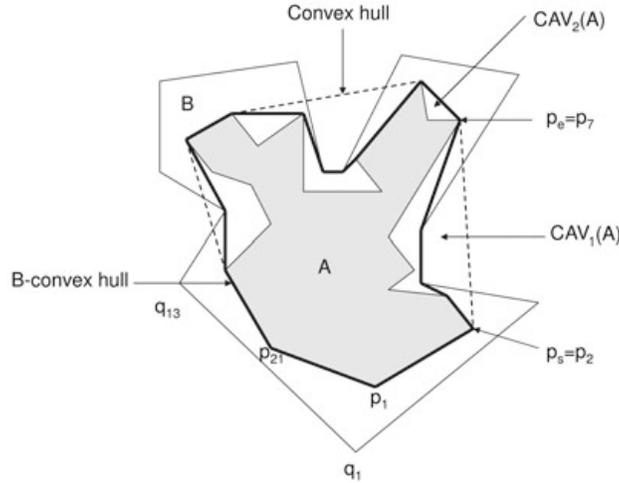
For example, in Fig. 1, the only cavity of  $A$  (defined by cover number 1), contains three non-connected subsets of cavities of  $B$ . A simple polygon  $A$  with  $n$  vertices has at most  $\lfloor n/2 \rfloor$  cavities; in the maximum case all the cavities are triangles. In general, cavities are notated by  $\text{CAV}_i(A)$ , where  $i = 1, 2, \dots$  follows the order of vertices on the frontier of  $A$ . A polygon is non-convex iff it has at least one cavity. A cavity is again a simple polygon with a minimum of three vertices, bounded by three straight line segments or more. This polygon may be convex or non-convex.

**Definition 2.** A cover is a straight line segment in the frontier of  $\text{CH}(A)$  that is not part of the frontier of  $A$ .

A cover separates a uniquely assigned cavity from the exterior of  $\text{CH}(A)$ . A cover is defined by two vertices  $p_s$  and  $p_e$ , the start vertex of a cavity and the end

---

<sup>1</sup> A Jordan curve separates two connected regions the plane, here the interior of the polygon from the exterior of the polygon.



**Fig. 2.** The inner polygon  $A$  has three cavities with nonempty intersections with cavities of the outer polygon  $B$ . Cavity  $CAV_2(A)$  has an empty intersection with cavities in  $B$ .

vertex of a cavity. The maximum depth of stacked cavities for the polygons  $A$  and  $B$  shown in Fig. 2 is two.

Let  $A$  be a simple polygon with  $n$  vertices,  $A = \langle p_1, p_2, \dots, p_n \rangle$ , and let  $B$  be a simple polygon with  $m$  vertices,  $B = \langle q_1, q_2, \dots, q_m \rangle$ , with  $A \subseteq B \subset \mathbb{R}^2$ .

**Definition 3.** A polygon  $A$  is  $B$ -convex iff any straight line segment in  $B$  that has both end points in  $A$ , is also contained in  $A$ . The convex hull of  $A$  relatively to  $B$  (in short, the  $B$ -convex hull of  $A$ ; formally  $CH_B(A)$ ) is the intersection of all  $B$ -convex polygons containing  $A$ .<sup>2</sup>

The *minimum length polygon* (MLP) of a 2D digital object coincides with the relative convex hull of an inner grid polygon relatively to an outer grid polygon, normally defined in a way like simulating an inner and outer Jordan digitization. In 2D digital imaging, an object is a digitization of a measurable set  $S \subset \mathbb{R}^2$  into a regular grid. We assume that a 2D picture  $P$  is composed of equally sized squares, where edges have length 1 and centers have integer coordinates. The inner polygon  $A$  is the union of all grid squares completely contained in the topological interior of a given object  $S \subseteq \mathbb{R}^2$ . The outer polygon  $B$  is the union of all grid squares having a nonempty intersection with the set  $S$  and the inner polygon  $A$ . The unknown frontier of the set  $S$  is assumed to be a Jordan curve  $\gamma$  located between the frontiers of polygons  $A$  and  $B$ .

The relative convex hull of  $A$  relatively to  $B$  (i.e., the MLP) is also located between the frontiers of polygons  $A$  and  $B$ , and its length is an approximation of the length of  $\gamma$  that converges to the true length of  $\gamma$  with increasing grid resolution [2]. It is uniquely defined for a given digitized object. The inner and the outer polygon of a Jordan digitization have the constraint that they are at

<sup>2</sup> This definition can also be generalized to higher dimensions.

Hausdorff distance 1. Mappings exist between vertices and between cavities in  $A$  and in  $B$ . The MLP may be calculated in linear time [1], where “linear” refers (e.g.) to the total number of grid squares between the inner and outer polygon.

In the general case, a simple polygon  $A$  inside of a simple polygon  $B$ , does not necessarily satisfy those constraints or properties, introduced due to the specifics of the regular grid. For the general case it is known that for all simple polygons  $A$  and  $B$ , only convex vertices of polygon  $A$  and only concave vertices of polygon  $B$  are candidates for the vertices of the relative convex hull [8].

Recently [5], an *arithmetic MLP* (AMLP) and a *combinatorial MLP* (CMLP) were proposed with the purpose of designing new algorithms for the computation of MLP.<sup>3</sup> For briefly discussing the AMLP, we consider the Gauss digitization of sets  $S \subset \mathbb{R}^2$ : All grid squares with their centroids in  $S$  belong to the resulting digital object. A set of connected grid squares is also called a *polyomino*. To be precise, a digital object is a polyomino iff it is 4-connected and the complement is 4-connected. The frontier of such a digitized object is a Jordan curve consisting of grid edges that separate the interior of the object from the exterior; these isothetic edges can be encoded by a Freeman chain code. The arithmetic definition is based on the fact that the tangential cover of the frontier of a digital object is a sequence of maximal digital straight lines (MDSS). The frontier of a digital region can always be uniquely divided into MDSS, assuming the start point is fixed (e.g., uppermost, leftmost). The object is *digitally convex* iff each pair of adjacent MDSSs of the tangential cover takes a convex turn. There are four different types of connected MDSSs with a single point overlap, called *zones*. A convex (concave) zone is an inextensible sequence of MDSSs where each pair of consecutive MDSSs takes a convex (concave) turn. The so-called *inflexion zones* are those parts where a convex turn is followed by a concave turn, or a concave turn is followed by a convex turn.

For a connected part of the frontier with only two kinds of steps (note: a digital straight line includes only two kinds of steps,  $c$  and  $c + 1 \pmod{4}$ ) the left envelope (resp. right envelope) is the sequence of straight lines of the convex hull of the associated vertices of the inner polygon (outer polygon). It is a polygonal line starting at  $v_i$  and ending at  $v_j$  clockwise (counterclockwise). The AMLP is a closed polygonal line defined by those zones that may be convex, non-convex, or a segment connecting a vertex of the outer polygon with a vertex of the inner polygon, or a segment connecting a vertex of the inner polygon with a vertex of the outer polygon. The CMLP is based on a combinatorial approach (for details see [5]). Both definitions are equivalent to the MLP, however they lead to different linear time algorithms.

---

<sup>3</sup> The authors of [5] motivate their work by a claim that the algorithm in [1] is faulty. Actually, it appears that the theoretical preparation of the algorithm in [1] in Euclidean space (when using Euclidean straight segments between vertices of the inner and outer polygon) is correct, but when using digital straight segments (DSSs) for an efficient implementation of the algorithm in the grid, only a simplified version of a DSS algorithm (DSSs up to complexity level 2 only) is used.

## 2.2 Algorithms for Calculating Relative Convex Hulls

Computing the relative convex hull of one simple polygon with respect to another simple polygon can be done in nearly (see below) linear time. In [11], the author changes the task to a shortest path problem between two vertices in a simple polygon by considering one extreme vertex of the inner polygon as start and end vertex at the same time, and by cutting  $B \setminus A$ , thus introducing a new (double-oriented) edge and combining outer and inner polygon into a single polygon. Then he proposes the triangulation of this resulting polygon  $B \setminus A$ , followed by finding the shortest path. The triangulation can be done in  $\mathcal{O}(n \log \log n)$  time [10], and finding the shortest path can be done in linear time, using this triangulation. However the triangulation is a very complex operation, and [11] provides no further details besides such a rough sketch.

The algorithm in [1] computes the MLP in linear time with respect to the total number of vertices. It traces the inner polygon (or outer polygon) counter-clockwise, saves the coordinates of all convex or concave vertices in a list, and it marks each vertex with a plus sign if it takes a positive turn (convex), and a minus sign if it takes a negative turn (concave). Collinear vertices are ignored because they are not candidates for the MLP. All concave vertices of  $A$  are replaced by the concave vertices of  $B$ , by applying the (existing!) bijective map between those vertices. This step can be done in linear time. In a second run, the algorithm starts at a known MLP-vertex and it computes the next vertex of the relative convex hull by checking its location between the negative sides (straight line between known MLP-vertex and next concave vertex) and the positive sides (straight line between known MLP-vertex and next convex vertex). This step needs only linear time because candidate vertices need only be checked once. However the number of computations could be reduced by copying those vertices into the final list of the relative convex hull that belong to the convex hull of the inner polygon. (See the footnote on the previous page.)

Linear time algorithms for the computation of AMLP and CMLP are given in [5]. The authors point out that those algorithms are simpler than existing ones, and that they are easier to implement. The algorithm for the computation of the AMLP computes first the tangential cover of a digitized object in linear time. The tangential cover is decomposed into zones (convex, concave, inflexion), and for each zone the vertices of the convex hulls of the polygonal line of the zone are computed using the algorithm in [4], and then added to a list. Both algorithms work only for polyominoes, and the number of different zones increases with the number of cavities.

## 3 Theoretical Results

We state the following for highlighting two obvious facts:

**Proposition 1.** *The  $B$ -convex hull of a simple polygon  $A$  is equal to the convex hull of  $A$  iff the convex hull of  $A$  is completely contained in  $B$  (i.e.,  $\text{CH}(A) \subseteq B$ ), or if  $B$  is convex.*

On the other hand, the B-convex hull of a simple polygon  $A$  is different to  $\text{CH}(A)$  if there exist at least one cavity in  $A$  and one cavity in  $B$  such that the intersection  $\text{CAV}_i(A) \cap \text{CAV}_j(B)$  is not empty.

**Theorem 1.** *All vertices of the convex hull of a simple polygon  $A$  inside a simple polygon  $B$  are vertices of the B-convex hull of  $A$ .*

*Proof.* First we consider start and end vertices  $p_s$  and  $p_e$  of one fixed cavity  $\text{CAV}(A)$ . They are per definition vertices of the convex hull of  $A$ . If the intersection  $\text{CAV}(A) \cap \text{CAV}(B)$  is empty then  $B$  has no vertices inside the cavity of  $A$ , and because  $A \subseteq B$ , the straight line between  $p_s$  and  $p_e$  connects vertices in  $A$  and  $B$ . Assuming that  $\text{CAV}(A) \cap \text{CAV}(B)$  is not empty then there is at least one vertex  $q_i \subseteq B$  inside the cavity of  $A$ , and the straight line between  $p_s$  and  $p_e$  crosses the exterior of  $B$ . A polygonal line between  $p_s$  and  $p_e$ , connecting the vertices of  $B$  inside the cavity, exists such that all of its line segments do not cross the frontier of  $A$ , and also do not cross the frontier of  $B$ .

Now we consider two consecutive vertices of  $\text{CH}(A)$  that are not the start or the end of a cavity; they belong to the  $\text{CH}_B(A)$  per definition.  $\square$

We assume that the convex hull of  $A$  is saved in a deque  $D(A)$  after the application of the Melkman algorithm. We can copy those vertices to the B-convex hull of  $A$ , and we have to insert additional vertices into the deque  $D(A)$  between  $p_s$  and  $p_e$ , for each cavity in  $A$ .

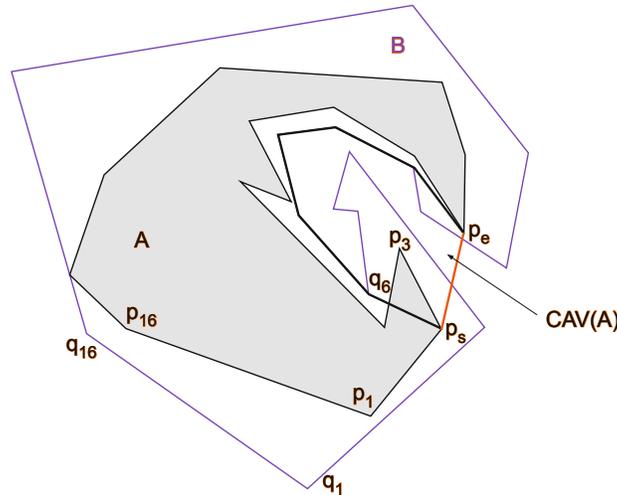
Let us consider cavities in polygons  $A$  and  $B$  with  $\text{CAV}(A) \cap \text{CAV}(B) \neq \emptyset$ . Let  $I_{new}$  be the polygon inside the cavity of  $A$  that is defined by vertices  $p_s$  and  $p_e$  in  $A$  and all the vertices in  $B$  located inside this cavity of  $A$  (see Fig. 3, with  $I_{new} = \langle p_s, q_3, q_4, \dots, q_{11}, p_e \rangle$ ).

**Theorem 2.** *All the vertices of the convex hull of  $I_{new}$  belong to the relative convex hull of  $A$  between  $p_s$  and  $p_e$ .*

*Proof.* We assume that  $I_{new}$  has no cavity. Then  $I_{new}$  is convex and all the straight lines between vertices of  $I_{new}$  are inside the convex hull of  $A$ , and they do not cross the exterior of  $B$ . They do not cross  $A$  because  $A \subseteq B$ . If  $I_{new}$  has a cavity with vertices of  $B$  inside the cavity then the vertices  $p_s$  and  $p_e$  on  $I_{new}$  belong obviously to the convex hull of this polygon. The straight lines between those vertices are completely in  $\text{CH}(A)$  and in  $B$  (see  $q_{10}, q_{11}, p_{10}$  in Fig. 3). If vertices of  $A$  are inside the cavity (see  $p_2, p_3, q_6$  in Fig. 3) then the straight line between vertices  $p_s$  and  $p_e$  would cross  $A$ . A polygonal line starting at  $p_s$  and ending at  $p_e$  connects vertices in  $A$ .  $\square$

If the new polygon  $I_{new}$  has a cavity  $\text{CAV}(I_{new})$  with vertices of  $A$  inside, then the start and the end vertex of  $\text{CAV}(I_{new})$  and the vertices of  $O_{new}$  constitute again a new inner polygon, and the start and the end vertex of  $\text{CAV}(I_{new})$  and the vertices of  $I_{new}$  constitute a new outer polygon (see Fig. 3). This establishes the basic idea of the new (recursive) algorithm.

We follow, in counterclockwise order, both frontiers of the original polygons. The computation of convex hulls starts usually (in computational geometry)



**Fig. 3.** Polygon  $A$  has one cavity.  $I_{new}$  has one cavity with one vertex of  $A$  inside.

at vertices with extreme values because it is obvious that vertices of  $A$  with extreme values (maximum and minimum  $y$ -coordinate, maximum and minimum  $x$ -coordinate) belong to the convex hull  $CH(A)$ ; those extrema are also vertices of the relative convex hull  $CH_B(A)$ . Furthermore, for the computation of the convex hull and for finding overlapping cavities in  $A$  and in  $B$ , we use the fact that a vertex  $p_i$  of a polygon is convex if the frontier takes a positive turn, that means the determinant  $t(p_{i-1}, p_i, p_{i+1}) > 0$ , and analog a vertex is concave if the value of the determinant is negative. A vertex is collinear if the value of the determinant is zero.

In the digital image case, there exists a bijective mapping between cavities in  $A$  and in  $B$  for the inner and outer Jordan digitization polygons. This mapping is a useful constraint for the computation of the MLP. It simplifies the process of finding overlapping cavities in  $A$  and  $B$  in this special situation.

## 4 Recursive Algorithm

Our new algorithm is a recursive procedure. The base case of the recursion, where it stops, is the triangle. We use Theorem 1: The relative convex hull  $CH_B(A)$  for simple polygons  $A \subseteq B$  is only different from  $CH(A)$  if there is at least one cavity in  $A$  and one in  $B$  such that the intersection of those cavities is not empty. The algorithm copies vertices of the convex hull of the inner polygon one by one until it finds a cavity. If it detects a cavity in  $A$ , then it finds the next cavity in  $B$  that has a nonempty intersection with the cavity in  $A$ , if there is any. The algorithm computes the convex hull of the new inner polygon  $I_{new}$  (all vertices of  $B$  inside the cavity of  $A$ , also the start vertex of  $CAV(A)$ , and also the end vertex of  $CAV(A)$ ). For each cavity in this convex hull, the algorithm computes the convex hulls of the next new polygons. If there is no cavity

remaining, then it inserts the computed vertices inside the cavity, between  $p_s$  and  $p_e$ , for all cavities in  $A$ , and it returns the relative convex hull of the given polygon  $A$  relatively to  $B$ . - This is the basic outline, and we discuss it now more in detail.

We assume that vertices of  $A$  and  $B$  are given with their coordinates in a list. We compute the convex hulls of both polygons by applying the Melkman algorithm [4]. The computation for both polygons starts at the vertices with minimum  $y$ -coordinates. For a given ordered set of  $n$  vertices  $A = \langle p_1, p_2, \dots, p_n \rangle$ , the algorithm delivers the convex hull in a deque  $D(A)$  where the first and the last element are the same vertices.

The difference between the indices of two consecutive vertices  $p_j$  and  $p_i$  in the resulting deque  $D(A)$  is equal to 1 if there is no cavity between  $p_j$  and  $p_i$ . We do not change  $D(A)$ . A cavity in  $A$  with the starting vertex  $p_s = p_j$  and the end vertex  $p_e = p_i$  has been found if  $(i - j) > 1$ . The next step finds a cavity in the convex hull of the outer polygon. It searches the deque  $D(B)$ . If it finds a cavity, it checks if a straight line between the vertices of  $B$  crosses the straight line  $p_s$  and  $p_e$ , and it computes the convex hull for  $p_s$ ,  $p_e$  and all the vertices of  $B$  that are left of  $\overline{p_s p_e}$  and inside the cavity. If all vertices inside a cavity of  $B$ , including  $q_s$  and  $q_e$ , are on the right of straight line  $\overline{p_s p_e}$ , then this cavity of  $B$  does not intersect a cavity of  $A$ ; see  $CAV_2(A)$  in Fig. 2. The relative convex hull does not change between  $p_s$  and  $p_e$ . We check the next cavity of  $B$ , with  $q_s$  on the right of  $\overline{p_s p_e}$ . The convex hull changes if  $q_s$  is on the right of  $\overline{p_s p_e}$  and if one vertex between  $q_s$  and  $q_e$ , saved in the original list  $B$ , is on the left of  $\overline{p_s p_e}$ .

For example, consider Fig. 3. In this example the convex hull of  $A$  is saved in a deque and the set of vertices equals  $D(A) = \langle p_1, p_2, p_{10} \dots p_{16}, p_1 \rangle$ . We trace the deque. The difference between the first two indices is 1, we calculate the difference between the second and the third vertex. Between  $p_2 = p_s$  and  $p_{10} = p_e$  there must be a cavity because the difference of the indices is larger than 1. Vertices of  $B$  inside the cavity between  $p_s$  and  $p_e$  define now the new inner polygon with vertices  $I_{new} = \langle p_s, q_3, q_4, \dots, q_{11}, p_e \rangle$ , and vertices of  $A$  inside the cavity between  $p_s$  and  $p_e$  define a new outer polygon with vertices  $O_{new} = \langle p_s, p_3, p_4, \dots, p_9, p_e \rangle$ . All vertices of the convex hull  $D(I_{new}) = \langle p_s, q_6, q_7, \dots, q_{10}, p_e, p_s \rangle$  are vertices of the relative convex hull. The polygon  $I_{new}$  has one cavity  $p_2, p_3, q_6$  with one vertex of polygon  $A$  inside. This defines again a new inner polygon. The convex hull of three vertices is always the same set of three vertices. The recursion stops. We replace  $p_2$  and  $q_6$  with  $p_2, p_3, q_6$  in  $D(I_{new})$ . We continue to check for cavities until we reach  $p_e$ . The adjusted deque replaces the start and the end vertices in  $D(A)$ . In our example (see Fig. 3) the next cavity in  $D(I_{new})$  starts at  $q_{10}$  and ends at  $p_e$ . But inside the cavity there is no element of the outer polygon. Thus, the relative convex hull does not change.

We continue to trace the deque  $D(A)$  at  $p_{10}$  until  $p_1$  is reached, and we skip vertex by vertex because there is no other cavity in  $A$ ;  $D(A)$  stays unchanged. This concludes the example.

The following pseudo code provides the basic structure of the algorithm.

**Algorithm 1.** (Calculation of the relative convex hull)

*Input:* Simple polygons  $A = \langle p_1, p_2, \dots, p_n \rangle$  and  $B = \langle q_1, q_2, \dots, q_m \rangle$ ,  $A \subseteq B$ .

*Output:* Relative convex hull in  $D(I)$  (counterclockwise).

- 1: Initialize  $D(I) = \emptyset$ ,  $D(O) = \emptyset$
- 2: Call Procedure  $\text{RCH}(A, B, D(I), D(O), n, m, p_1, p_n)$

Note that  $p_1 = p_{n+1}$  and  $q_1 = q_{m+1}$ . This algorithm applies recursively a procedure  $\text{RCH}(I, O, D(I), D(O), l, t, p_s, p_e)$  that is sketched in Fig. 4.

- 1: Compute convex hulls of  $I$  and  $O$  in deques  $D(I)$  and  $D(O)$ , respectively,  $l$  is number of vertices in  $I$ ,  $t$  is number of vertices in  $O$ ,  $p_s$  is the start vertex and  $p_e$  is the end vertex of the inner polygon.  $k = 1$  and  $j = 1$  are loop variables.
- 2: Remove the last elements in  $D(I)$  and  $D(O)$
- 3: **while**  $k < l$  **do**
- 4:   **if** Cavity between two consecutive vertices in  $D(I)$ ,  $(d_k$  and  $d_{k+1})$  **then**
- 5:      $p_s = d_k$  and  $p_e = d_{k+1}$
- 6:     **while**  $j < t$  **do**
- 7:       **if** Overlapping cavity between two consecutive vertices in  $D(O)$  **then**
- 8:         Update  $I$  such that  $I$  includes  $p_s$  and  $p_e$  and all vertices in  $O$  inside the cavity of  $I$ ,  $L$  is the number of vertices
- 9:         **if**  $L > 3$  **then**
- 10:         Update  $O$  such that  $O$  includes  $p_s$  and  $p_e$  and all vertices in  $I$  inside the cavity of  $I$ ,  $T$  is the number of vertices
- 11:         Call  $\text{RCH}(I, O, D(I), D(O), L, T, p_s, p_e)$
- 12:         **end if**
- 13:         Insert  $q$  between  $p_s$  and  $p_e$  in  $D(I)$
- 14:         **end if**
- 15:         **end while**
- 16:         Return  $D(I)$
- 17:         **end if**
- 18:     **end while**
- 19:     Return  $D(I)$

**Fig. 4.** Recursive procedure

The Melkman algorithm delivers the convex hull in a deque with the first element at the bottom of the deque and also at the top of the deque. We need to remove the elements from the top after the computation of convex hulls.

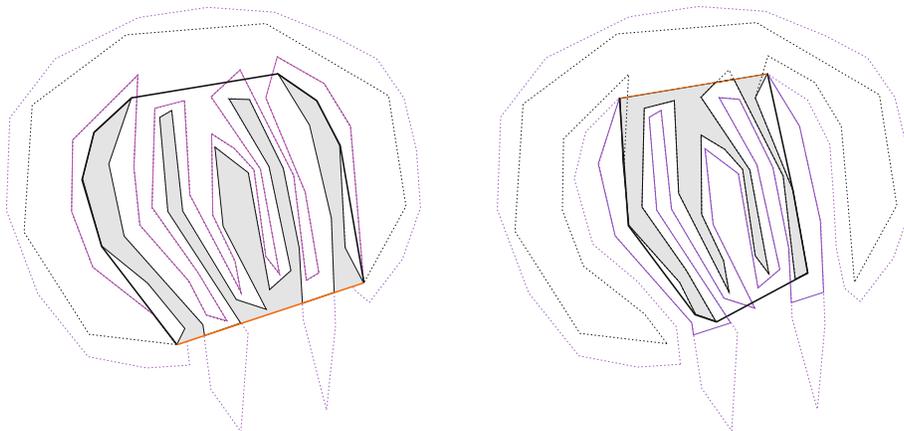
## 5 Discussion

Note that any vertex on  $A$  or  $B$  is accessed by the algorithm only at most as often as defined by the depth of a stacked cavity. This defines this algorithm as being of linear time complexity, measured in the total number of vertices on  $A$  and  $B$ , if (!) this depth is limited by a constant, but of quadratic time in the worst

case sense. For the general case, if there are “many” cavities in  $A$ , then they are all “small”, and the algorithm has again linear run-time behavior. If there is just one “big” cavity (similar to Fig. 3), then the recursion only proceeds for this one cavity, and we are basically back to the upper bound for the original input, but now for a reduced number of vertices. The worst case in time complexity is reached if there is a small number of “large cavities” in  $A$  and  $B$ , causing repeated recursive calls within originally large cavities of  $A$ , and the number of recursive calls defines the *depth* of those stacked cavities. However, such cases appear to be very unlikely in applications.

We discuss a few more details briefly for the example shown in Fig. 1. The cover  $\overline{p_s p_e}$  of the first cavity cuts off three components of the original polygon  $B$ . The resulting simple polyline (from  $p_s$  to  $p_e$ ) is not allowed to cross the segment  $\overline{p_s p_e}$ , see Fig. 5, left. We have two simple polylines from  $p_s$  to  $p_e$ , defining the inner and the outer polygon. Note that the inner polygon is now in general not simple, but this is not restricting the algorithm, because double edge orientations can only occur on  $\overline{p_s p_e}$  (and the Melkman algorithm is for simple polylines anyway). For the second cavity, see Fig. 5, right. Here, the bold black line shows the calculated convex hull of the inner simple polyline, which defines again a new cavity.

Now consider the special case that  $A$  and  $B$  may be considered to be inner and outer digitizations of a set  $S \subset \mathbb{R}^2$ . The inner and outer polygons of a Jordan digitization satisfy special constraints. The step of finding the cavities with a nonempty intersection is faster because the algorithm can now use the existing [1] bijective map between cavities. Once we have found a cavity in  $A$  then there must be a cavity in  $B$ , and vertices for the new inner polygon are easy to find. First, the algorithm traces the inner polygon and the outer polygon counterclockwise and it saves the coordinates of all convex or concave vertices in a list. Then the recursive procedure returns the MLP. The maximum number of cavities in a polyomino with  $n$  convex or concave vertices is  $\lfloor n/2 \rfloor$ . In this maximum number case, the algorithm would stop after two loops.



**Fig. 5.** First and second cavity for the example in Fig. 1

## 6 Conclusion

We presented a completely new algorithm for the computation of the  $B$ -convex hull of arbitrary simple polygons. It is a recursive procedure that is very simple and of low time complexity. The procedure uses a linear time algorithm for the computation of convex hulls, both for inner and outer polygons, starting with the given input polygons, and then continuing with the polygons defined in recursively defined cavities. The algorithm runs in linear time if the maximum depth of stacked cavities of  $A$  is limited by a constant. We continue to study the expected time complexity of the algorithm under some general assumptions of variations (i.e., distribution) for possible input polygons  $A$  and  $B$ .

## References

1. Klette, R., Kovalevsky, V.V., Yip, B.: Length estimation of digital curves. *Vision Geometry*, SPIE 3811, 117–129 (1999)
2. Klette, R., Rosenfeld, A.: *Digital Geometry*. Morgan Kaufmann, San Francisco (2004)
3. Klette, R., Zunic, J.: Multigrid convergence of calculated features in image analysis. *J. Mathematical Imaging Vision* 13, 173–191 (2000)
4. Melkman, A.: On-line construction of the convex hull of a simple polygon. *Information Processing Letters* 25, 11–12 (1987)
5. Provençal, X., Lachaud, J.-O.: Two linear-time algorithms for computing the minimum length polygon of a digital contour. In: Brlek, S., Reutenauer, C., Provençal, X. (eds.) *DGCI 2009*. LNCS, vol. 5810, pp. 104–117. Springer, Heidelberg (2009)
6. Sklansky, J.: Measuring cavity on a rectangular mosaic. *IEEE Trans. Computing* 21, 1355–1364 (1972)
7. Sklansky, J., Kibler, D.F.: A theory of nonuniformly digitized binary pictures. *IEEE Trans. Systems, Man, and Cybernetics* 6, 637–647 (1976)
8. Sloboda, F., Stoer, J.: On piecewise linear approximation of planar Jordan curves. *J. Comput. Appl. Math.* 55, 369–383 (1994)
9. Sloboda, F., Zatko, B., Stoer, J.: On approximation of planar one-dimensional continua. In: Klette, R., Rosenfeld, A., Sloboda, F. (eds.) *Advances in Digital and Computational Geometry*, pp. 113–160 (1998)
10. Tarjan, R.E., Van Wyk, C.J.: An  $\mathcal{O}(n \log \log n)$  algorithm for triangulating a simple polygon. *SIAM J. Computing* 17, 143–178 (1988)
11. Toussaint, G.T.: An optimal algorithm for computing the relative convex hull of a set of points in a polygon. In: *EURASIP, Signal processing III: Theories and Applications, Part 2*, pp. 853–856. North-Holland, Amsterdam (1986)

# Apéndice C

ARTÍCULO DE A. MELKMAN



## ON-LINE CONSTRUCTION OF THE CONVEX HULL OF A SIMPLE POLYLINE

Avraham A. MELKMAN \*

*Ben Gurion University of the Negev, Beer Sheva, Israel*

Communicated by Alan Shaw

Received 2 April 1985

Revised 27 November 1985 and 9 September 1986

*Keywords:* Convex hull, simple polygon, analysis of algorithms

### 1. Introduction and description of the algorithm

After McCallum and Avis [4] showed that the convex hull of a simple polygon  $P$  with  $n$  vertices can be constructed in  $O(n)$  time, several authors [1,2,3] devised simplified algorithms for this problem. Graham and Yao [2] presented a particularly simple and elegant one. After finding two points of the convex hull, their algorithm generated all other hull vertices using only one stack for intermediate storage. It is the purpose of this short article to show that a slightly modified version of their algorithm constructs, on-line, the convex hull of any simple polyline in  $O(n)$  time. In contrast, the on-line construction of a nonsimple polyline requires  $O(n \log n)$  time, as shown by Preparata [5]. In the special case of a simple polygon our algorithm produces the convex hull without first identifying two of the hull vertices, as was required in [2]. The price that we pay is the use of a deque instead of a queue. After this article was submitted, we learned of a similar approach taken by Tor and Middleditch [6], who embed it in an algorithm for the convex decomposition of a simple polygon.

To keep this article short, we use where possible the definitions and notations of Graham and Yao. The polyline  $P = \langle v_1, \dots, v_m \rangle$  is assumed to

be given by an input list of its vertices as they are encountered in an ordered traversal, and the algorithm will consider the vertices in that order. For an ordered triple of points  $x, y, z$ , the function  $(x, y, z)$  assumes values 1, 0, or  $-1$  depending on whether  $z$  is to the right of, collinear with, or to the left of the directed line from  $x$  to  $y$ .

The main data structure used is a deque of vertices,  $D = \langle d_b, \dots, d_t \rangle$ , where the variable  $b$  points to the bottom of the deque and  $t$  to its top, and  $d_b$  and  $d_t$  will always refer to the same vertex. Thus,  $D$  describes a closed curve; the polygon enclosed by this curve will also be referred to as  $D$ . Variable  $v$  refers to the input vertex under consideration. 'Pushing  $v$ ' means executing  $[t \leftarrow t + 1; d_t \leftarrow v]$ , 'popping  $d_t$ ' means setting  $[t \leftarrow t - 1]$ , 'inserting  $v$ ' means executing  $[b \leftarrow b - 1; d_b \leftarrow v]$ , and 'removing  $d_b$ ' means setting  $[b \leftarrow b + 1]$ . The algorithm halts when its input is exhausted.

#### Algorithm Hull

- $t \leftarrow -1; b \leftarrow 0;$   
 $v_1 \leftarrow \text{input}; v_2 \leftarrow \text{input}; v_3 \leftarrow \text{input};$   
**if**  $(v_1, v_2, v_3) > 0$   
    **then begin** push  $v_1$ ; push  $v_2$ ; **end**  
    **else begin** push  $v_2$ ; push  $v_1$ ; **end**  
    push  $v_3$ ; insert  $v_3$ ;
- $v \leftarrow \text{input};$   
**until**  $(v, d_b, d_{b+1}) < 0$  **or**  $(d_{t-1}, d_t, v) < 0$   
    **do**  $v \leftarrow \text{input}$  **end**;

\* Present affiliation: Department of Computer Science, Columbia University, New York, NY 10027, U.S.A.

3. **until**  $(d_{t-1}, d_t, v) > 0$  **do** pop  $d_t$  **end**;  
push  $v$ ;
4. **until**  $(v, d_b, d_{b+1}) > 0$  **do** remove  $d_b$  **end**;  
insert  $v$ ;  
**goto** 2.

## 2. Correctness of Algorithm Hull

In this section we prove the following theorem.

**Theorem.** *Algorithm Hull finds the convex hull of P correctly and in linear time.*

**Proof.** Evidently, the algorithm is linear in the number of vertices of P, as each vertex is pushed (inserted) at most once and popped (removed) at most once. We use the following characterization of the convex hull H of P:

- (i) H is convex,
- (ii) P is contained in H,
- (iii) the set of vertices of H is a subset of the set of vertices of P.

The last property is built in to the algorithm. Thus, we will prove by induction that the following hypothesis is true each time a new vertex is read in step 2:

- (H) The deque  $D = \langle d_b, \dots, d_t \rangle$ , considered as a polygon, is convex and contains that part of the polyline P seen so far.

First of all, observe the following.

**Claim.** *All vertices rejected in step 2 are in, or on, the current convex hull contained in D.*

Indeed, if a vertex  $v$  is rejected, it is because both  $(d_b, d_{b+1}, v) \geq 0$  and  $(d_{t-1}, d_t, v) \geq 0$ , i.e.,  $v$  is to the right of both the edges  $\langle d_{t-1}, d_t \rangle$  and  $\langle d_b, d_{b+1} \rangle$ . Moreover, the polyline connects  $v$  with  $d_t$  and, because it is simple, does not cross the part of the polyline connecting  $d_{b+1}$  and  $d_{t-1}$ . Hence,  $v$  must lie within the convex polygon described by D.

Step 1 ensures that (H) holds when three vertices have been read (assuming they are not collinear). Suppose then inductively that (H) holds when  $k$  vertices have been read. Because of the Claim it may be assumed that  $v$  satisfies either  $(d_b, d_{b+1}, v) < 0$  or  $(d_{t-1}, d_t, v) < 0$  (or both), and that steps

3 and 4 are executed immediately after  $v$  is read and before the next vertex is read.

Suppose that, before execution of steps 3 and 4, the deque is  $D = \langle d_b, \dots, d_t \rangle$  and that after their execution the deque is  $D' = \langle d_k, \dots, d_m \rangle$ , with  $v = d_k = d_m$ . To prove  $D'$  to be convex we show that it describes a closed simple curve and that  $(d_i, d_{i+1}, d_{i+2}) > 0$ ,  $i = k, \dots, m-2$ , and  $(d_{m-1}, d_m, d_{k+1}) > 0$ . Because D is simple,  $D'$  can only be nonsimple if the edges  $\langle v, d_{k+1} \rangle$  and  $\langle d_{m-1}, v \rangle$  intersect edges of D; this cannot be the case since the polyline P is simple. Now,  $(d_i, d_{i+1}, d_{i+2}) > 0$ ,  $i = k, \dots, m-2$ , is either already true for D or else it is the result of steps 3 and 4. Suppose then by contradiction that  $(d_{m-1}, d_m, d_{k+1}) \leq 0$ . Then,  $v = d_m$  is on or to the right of the directed line from  $d_{m-1}$  to  $d_{k+1}$ , and at the same time it must be to the right of the polyline  $\langle d_{k+1}, \dots, d_{m+1} \rangle$ . Thus,  $v$  is within D, a contradiction.

Finally, it is easily seen by induction that P is contained in  $D'$ , because it is contained in D and each time a vertex  $d_j$  is popped, the polygon  $\langle v, d_i, \dots, d_j, v \rangle$  is contained within the polygon  $\langle v, d_i, \dots, d_{j+1}, v \rangle$ ; similarly, the removal of a vertex only enlarges the polygon.  $\square$

## Acknowledgment

Karl Abrahamson and David Kirkpatrick devised a similar algorithm as part of their refereeing. The helpful suggestions of an anonymous referee, which simplified the algorithm and its proof, are also gratefully acknowledged.

## References

- [1] B.K. Bhattacharya and H. El Gindy, A new linear convex hull algorithm for simple polygons, IEEE Trans. Inform. Theory IT-30 (1984) 85-88.
- [2] R.L. Graham and F.F. Yao, Finding the convex hull of a simple polygon, J. Algorithms 4 (1984) 324-331.
- [3] D.T. Lee, On finding the convex hull of a simple polygon, Internat. J. Comput. Inform. Sci. 12 (1983) 87-98.
- [4] D. McCallum and D. Avis, A linear algorithm for finding the convex hull of a simple polygon, Inform. Process. Lett. 9 (1979) 201-206.
- [5] F.P. Preparata, An optimal real time algorithm for planar convex hulls, Comm. ACM 22 (1979) 402-405.
- [6] S.B. Tor and A.E. Middleditch, Convex decomposition of simple polygons, ACM Trans. Graphics 3 (1984) 244-265.

# Bibliografía

- [1] Diez-Canedo, J.M.: Fundamentos de teoría de optimización, Limusa, México (1987)
- [2] Klette, G.: Recursive Calculation of Recursive Convex Hulls. In: Proceedings DGCI, LNCS, vol. 6607 pp. 260-271. Springer, Heidelberg (2011)
- [3] Klette, R., Kovalevsky, V., Yip, B.: On the length estimation of digital curves. In: SPIE Vision Geometry VII, vol. 3811, pp. 52-63 (117-129) (1999)
- [4] Klette, R.: Multigrid convergence of geometric features. In: Bertrand, G. et al (Eds.), Digital and Image Geometry: Advanced Lectures, LNCS 2243, pp. 314-333, Springer, Berlin (2004)
- [5] Klette, R., Rosenfeld, A.: Digital Geometry - Geometric Methods for Digital Picture Analysis, Morgan Kaufmann Publ., Elsevier, USA (2004)
- [6] Klette, R., Yip, B.: Evaluation of curve length measurements, 15th Int. Conf. on Pattern Recognition, Vol. 1 (2000), pp. 1610-1614, and: The length of digital curves, Machine Graphics Vision, 9:673-703 (2000)
- [7] Li, F., Klette, R.: Euclidean Shortest Paths, Exact or Approximate Algorithms, Springer, London (2011)
- [8] Melkman, A.: On-line construction of the convex hull of a simple polyline, Information Processing Letters 25, pp. 11-12, North-Holland (1987)
- [9] Munkres, J.R.: Topology, 2nd edition, pp. 390, Prentice Hall, USA (2000)
- [10] Naber, G. L.: Topological methods in euclidean spaces, Dover publications, Inc., New York (1980)
- [11] O'Rourke, J.: Computational Geometry in C, Cambridge University Press, 2nd edition, USA (1998)
- [12] Sloboda, F., Stoer, J.: On piecewise linear approximation of planar Jordan curves, Journal of Comp. and Appl. Mathematics 55, pp. 369-383 (1994)
- [13] Sloboda, F., Zatco, B., Stoer, J.: On approximation of planar one-dimensional continua. In: Klette, R., Rosenfeld, A., Sloboda, F. (Eds.), Advances in Digital and Computational Geometry, pp. 113-160, Springer, Singapore (1998)

- [14] Toussaint, G.T.: An optimal algorithm for computing the relative convex hull of a set of points in a polygon. In: EURASIP, Signal Processing III: Theories and Applications, Part 2, pp 853-856, North-Holland (1986)
- [15] <http://cgm.cs.mcgill.ca/~athens/cs601/>  
*A history of linear-time convex hull algorithms for simple polygons*  
Aluopis, Greg  
Computational Geometry Lab  
School of Computer Science at McGill University.
- [16] <http://www.uni-kiel.de/psychologie/rexrepos/posts/diagBounding.html>  
*Convex hull, (minimum) bounding box, and minimum enclosing circle.*  
Christian-Albrechts-Universität zu Kiel
- [17] <http://lema.rae.es/drae/>  
*Definición de Algoritmo.*  
Real Academia Española (en línea)  
Vigésima segunda edición
- [18] <http://www.finestdaily.com/news/tech/japanese-robots-are-ready-to-run-the-marathon.html>  
*Japanese Robots are Ready to Run the Marathon*  
Lupica, George  
February 17, 2011
- [19] <http://www.whatsonxiamen.com/news17676.html>  
*Robovie PC, Vstone wins 1st robot marathon in Osaka, Japan*

# FIRMAS

---

Directora de tesis  
Dra. Petra Wiederhold Grauert  
Departamento de Control Automático

---

Dr. Gabriel Villa Salvador  
Departamento de Control Automático

---

Dr. Rubén Garrido Moctezuma  
Departamento de Control Automático

---

Dr. Feliú Sagols Troncoso  
Departamento de Matemáticas