UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

ILUMINACIÓN DE POLÍGONOS CON θ-MÓDEMS

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

RICARDO LÓPEZ LÓPEZ

TUTORA:
DRA. ADRIANA RAMÍREZ VIGUERAS

FACULTAD DE CIENCIAS
UNAM

2017

CIUDAD UNIVERSITARIA, CDMX

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Tesis: Iluminación de polígonos con θ-módems

Agosto 2017

"Las Matemáticas contribuyen a crear el necesario rigor en el pensar, a forjar una lógica exacta, base de un pensamiento ordenado; y aunque no son la panacea, quien se educa en esa lógica se resiste hasta orgánicamente a aceptar que se le mienta impunemente, que se distorsione el razonamiento ..."

Abel Peréz Zamorano

Índice general

Capítulo 1

Introducción

En una galería de arte es importante mantener vigiladas las obras expuestas debido a su valor. Para reducir los gastos originados por la vigilancia, es necesario ocupar la menor cantidad de guardias posibles sin dejar alguna obra descuidada. A esto se le conoce como *problemas de la galería de arte*, propuesto por primera ocasión en el año de 1973 por *Victor Kleen*, quien formuló la siguiente pregunta:

"*¿Cuántos guardias necesitamos para vigilar una galería de arte?*"

Posteriormente en 1975 *Chvátal* prueba que $\lfloor \frac{n}{3} \rfloor$ guardias son siempre suficientes y a veces necesarios para vigilar un polígono simple con n vértices [12]. Los problemas de la galería de arte han sido de gran interés en la geometría computacional y la combinatoria. Después de la aparición del artículo de *Chvátal*, diversas variaciones del problema han aparecido en la literatura, entre éstas encontramos: vigilar polígonos ortogonales, polígonos con hoyos, usar guardias móviles, guardias con visibilidad acotada, entre otras [25, 29, 32]. Recientemente *O. Aichholzer*, *R. Fabila-Monroy*, *D. Flores-Peñaloza*, *T. Hackl*, *J. Urrutia* y *B. Vogtenhuber* plantearon una nueva variante, llamado el *problema de iluminación con módems* [**?**]. El cual utiliza módems para *iluminar* el interior de un polígono, la diferencia entre un guardia y un módem es simple, la señal inalámbrica de un módem puede atravesar cierta cantidad de paredes y la visibilidad de un guardia no.

Los problemas de galería de arte son importantes en ciertas aplicaciones de las ciencias de la computación, debido a que estos son básicamente problemas de visibilidad y esta es de vital importancia en muchas aplicaciones computacionales; por ejemplo, algunas áreas de aplicación son: robótica, planeación de movimientos, visión computacional, graficación por computadora, comunicaciones, sistemas de información geográfica, entre otras [26, 36, 11, 34, 35, 45].

El constante avance en las tecnologías inalámbricas y el acceso prácticamente inmediato a una vasta fuente de información como lo es el Internet, han hecho que el uso de dispositivos móviles con acceso a redes inalámbricas se vuelva una necesidad para las

nuevas generaciones. Los polígonos son una buena aproximación de los planos de una construcción, por lo que el estudio del problema de iluminación con módems es de gran importancia para las ciencias de la computación. Dado lo anterior, en este trabajo además de mostrar un panorama general del área, se da apoyo a la docencia e investigación, implementando una herramienta que sea útil para el estudio de estos problemas.

1.1. Objetivos

- Mostrar las principales técnicas y resultados empleados en los teoremas de iluminación de galerías de arte en dos dimensiones.
- Mostrar los principales resultados obtenidos hasta el momento en las variantes del problema de la galería de arte: el problema de iluminación con módems y el problema de iluminación con reflectores.
- Diseñar e implementar algoritmos interactivos auxiliares en la construcción de polígonos simples, monótonos, ortogonales y monótonos-ortogonales.
- Implementar los algoritmos para construir el polígono de visibilidad de un punto, un k-módem y un θ_k-módem, en el interior de un polígono simple sin hoyos [29, 4]. Usar dichos algoritmos para desarrollar una aplicación interactiva, que ayude a visualizar el comportamiento de dichos objetos, dentro de polígonos simples, monótonos, ortogonales y monótonos-ortogonales.

1.2. Organización del trabajo

En la literatura existe una gran cantidad de artículos sobre problemas de la galería de arte, por lo que en este trabajo se realizó una breve recopilación de los artículos, de tal forma que se muestran en orden cronológico; algunos de los resultados obtenidos en el problema de iluminar la galería de arte; y en las variantes de iluminación con módems y con reflectores; así como las técnicas y herramientas básicas para sus pruebas.

Este trabajo está organizado de la siguiente manera:

El capítulo 2 presenta los conceptos básicos para la comprensión del los problemas de la galería de arte. Estos conceptos se pueden consultar ampliamente en [15] y [44].

En el capítulo 3 se muestra el proceso de diseño del conjunto de algoritmos interactivos para la construcción de polígonos simples, monótonos, ortogonales y ortogonales-monótonos. Además presenta el pseudocódigo y la complejidad de cada algoritmo.

En el capítulo 4 se exponen algunos de los teoremas de galería de arte para polígonos simples, ortogonales y polígonos con hoyos y se incluye el teorema clásico de *Chvátal*. Se concluye con una tabla que muestra un resumen de las complejidades de los resultados

mostrados.

En el capítulo 5 se presentan algunos de los resultados de los problemas de iluminación de polígonos usando reflectores y utilizando módems. Se explica el algoritmo de *A. L. Bajuelos*, *S. Canales*, *G. Hernández* y *A. M. Martins* [4], para calcular el polígono de visibilidad de un k-módem en un polígono simple sin hoyos. También se introduce el concepto de θ_k-módem y el problema de iluminación con θ_k-módems.

El capítulo 6 describe la aplicación *LitPolygons*, resultado de la implementación de los algoritmos mostrados en el capítulo 3 y en la sección 5.2.2. Se muestran la interfaz de usuario y sus componentes, algunos ejemplos de las tareas que puede realizar la aplicación, así como los requerimientos y tecnologías usadas en el desarrollo de ésta. Por último se explica su instalación en dispositivos móviles Android y en equipos de cómputo Mac y Linux (Ubuntu).

Capítulo 2

Conceptos fundamentales

En el área de la geometría computacional, los polígonos son de los objetos geométricos más utilizados para el diseño de soluciones a problemas de dicha área; los polígonos son una representación muy conveniente para varios objetos del mundo real, tal es el caso de las galerías de arte. En las siguientes secciones se definirán algunos conceptos básicos y propiedades de los polígonos que se estudian en este trabajo. Estas definiciones pueden encontrarse en los libros de geometría computacional tales como [15] y [44].

2.1. Definiciones

Definición 1. Un **segmento de línea** $s = \overline{ab}$ es un subconjunto cerrado de una línea contenido entre dos puntos a y b, a los cuales llamamos puntos extremos. Ver figura 2.1.

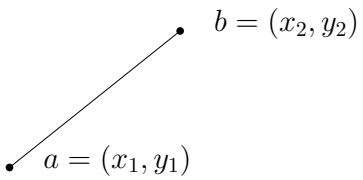


Figura 2.1: Segmento con puntos extremos a, b.

Definición 2. Un **polígono** P es la región acotada por una secuencia de n puntos en el plano, $v_1, v_2, \cdots, v_n$, con $n \geq 3$, llamados vértices de P, junto con el conjunto de segmentos de línea uniendo $\overline{v_i v_{i+1}}$, para $i = 1, 2, \cdots, n-1$ y $\overline{v_n v_1}$, llamados aristas de P. Ver figura 2.2.
El conjunto de aristas y vértices de P es llamado la *frontera* de P, denotada por δP y a la región acotada por δP se le nombra el *interior* de P.

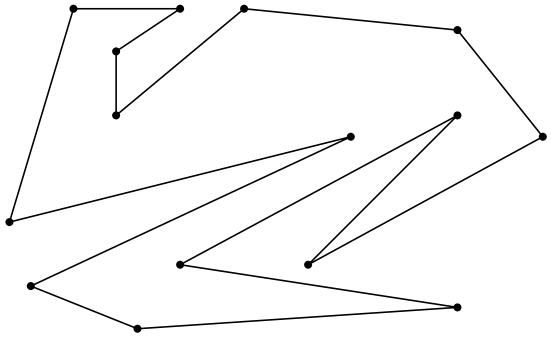
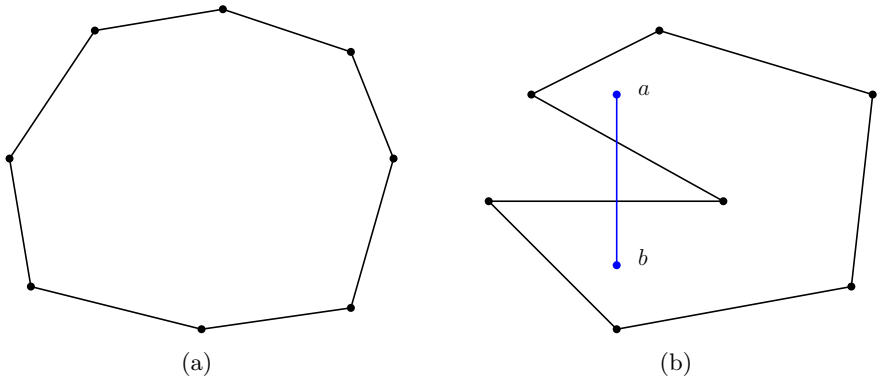


Figura 2.2: Polígono

Definición 3. Un polígono P es **convexo** si para cualquier par de puntos $a, b \in P$, el segmento de línea $s = \overline{ab}$ está contenido en el interior de P. Ver figura 2.3.



(a) (b)

Figura 2.3: (a) Polígono convexo (b) Polígono no convexo

Un polígono convexo tiene la propiedad de que todos los ángulos internos de sus vértices son menores o iguales que π.

Definición 4. Una **diagonal** de un polígono es un segmento de línea que conecta dos vértices de P, tal que está contenido en el interior de P y no toca a la frontera de P excepto en dichos vértices. Dos diagonales se intersectan si comparten un punto interior. Ver figura 2.4.

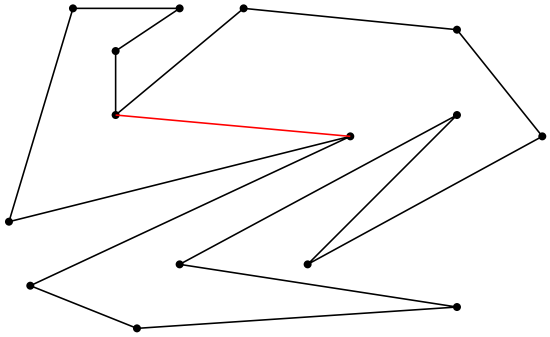


Figura 2.4: – Diagonal del polígono

Definición 5. Una **triangulación de un polígono** P es una descomposición de P en triángulos por un conjunto maximal de diagonales sin intersecciones. Ver figura 2.5.

Un conjunto de diagonales sin intersecciones es maximal si una nueva diagonal no puede ser agregada sin que se genere una intersección entre ellas.

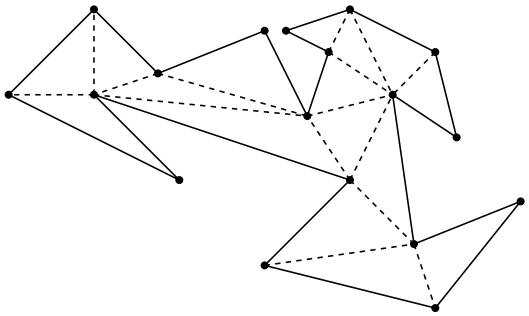


Figura 2.5: Triangulación de un polígono

Definición 6. Sea S un conjunto de puntos en el plano. S está en **posición general** si y sólo si no existen tres puntos en S colineales. Ver figura 2.6.

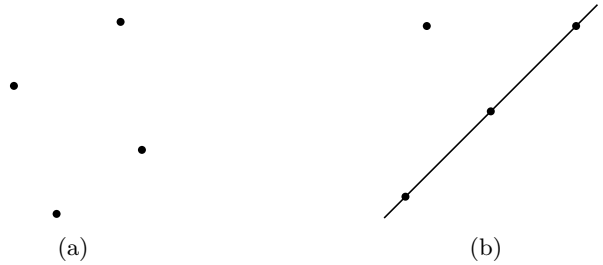


Figura 2.6: (a) Puntos en posición general (b) Puntos en posición no general

2.2. Polígono simple

Definición 7. Un polígono P es llamado **simple** si ninguna pareja de aristas no consecutivas se intersectan y no contiene ningún objeto en su interior. Ver figura 2.7.

Definición 8. Dado un polígono simple P y un conjunto de polígonos simples ajenos $P_1, P_2, \cdots, P_h$ contenidos en el interior de P, el conjunto $P - \{P_1 \cup P_2 \cup \cdots \cup P_h\}$ es llamado **polígono con hoyos**. En este caso decimos que P tiene h hoyos. Ver figura 2.7.

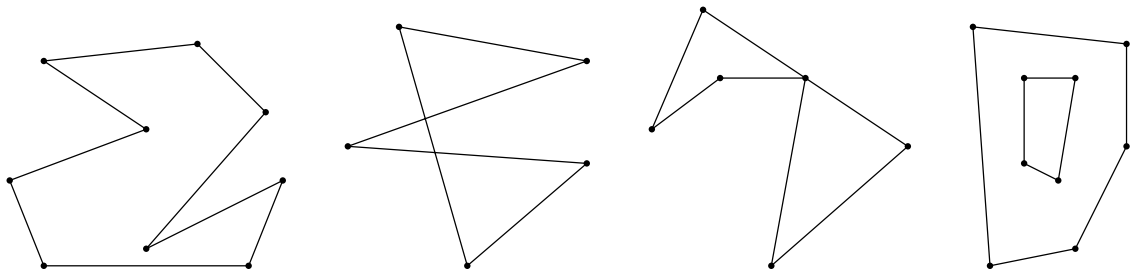


Figura 2.7: (a)Polígono simple, (b)-(d)Polígonos no simples, con (d) un polígono con 1 hoyo

2.3. Polígono monótono

Definición 9. Un polígono simple P es llamado **monótono** con respecto a una línea ℓ, si para cualquier línea ℓ' perpendicular a ℓ, la intersección de P con ℓ' es un segmento de línea, un punto o vacía. Ver figura 2.8.

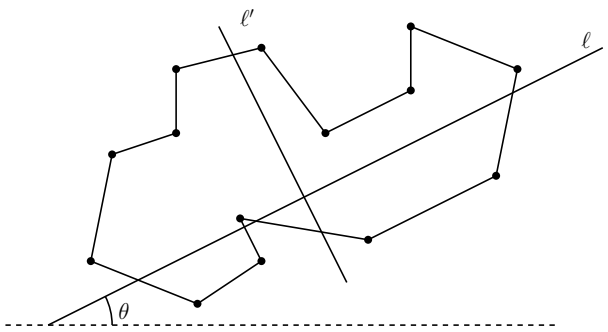


Figura 2.8: Polígono monótono

Un polígono que es monótono con respecto al eje X es llamado *x-monótono*, ver figura 2.9. La siguiente propiedad es característica de un polígono *x-monótono*: si caminamos sobre la frontera del polígono desde el vértice más a la izquierda hacia el vértice más a

la derecha, en cada posición el valor de la coordenada x no disminuye, es decir, no habrá retrocesos horizontales.

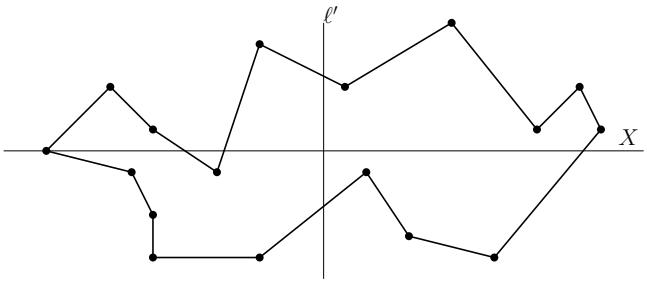


Figura 2.9: Polígono x-monótono

2.4. Polígono ortogonal

Definición 10. Un polígono simple es llamado **ortogonal** si todos sus lados (otra manera de referirse al conjunto de aristas) son paralelos a los ejes cartesianos, es decir, todos sus lados son paralelos al eje X o a el eje Y. Ver figura 2.10.

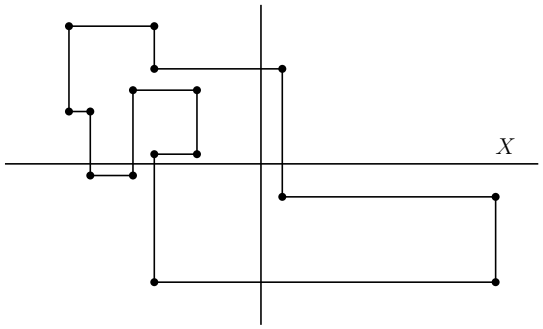


Figura 2.10: Polígono ortogonal

Es fácil ver que "*en realidad*" la mayoría de los edificios son ortogonales, por lo que estos polígonos resultan de especial interés en el estudio de problemas de iluminación y vigilancia.

2.5. Polígono ortogonal monótono

Definición 11. Un polígono **ortogonal monótono** conjunta las propiedades de los dos tipos de polígonos precedentes; es decir, es un polígono ortogonal y á su vez monótono. Ver figura 2.11.

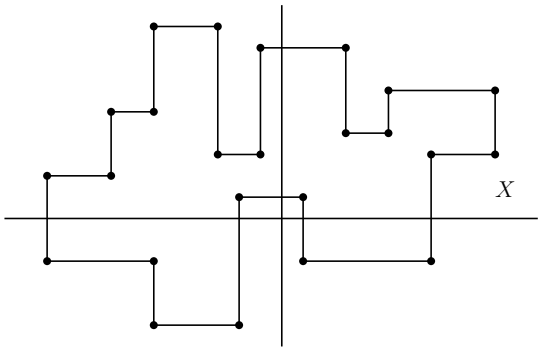


Figura 2.11: Polígono ortogonal x-monónoto

2.6. Gráficas

Las definiciones a continuación descritas pueden encontrarse en [7].

Definición 12. Una **gráfica** $G = (V, E)$ es un conjunto finito no vacío de elementos llamados vértices, denotado por V, junto con un conjunto de parejas de vértices distintos de G llamadas aristas, denotado por E. Ver figura 2.12.

Definición 13. Sean u y v dos vértices de $G = (V, E)$ diremos que son **adyacentes** si la pareja $\{u, v\} \in E$. Ver figura 2.12.

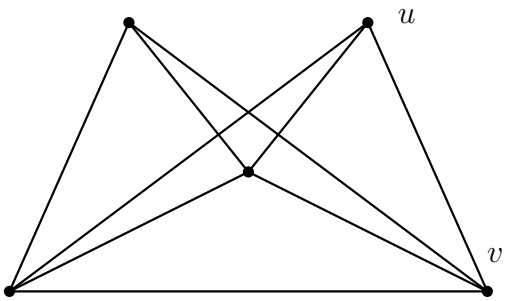


Figura 2.12: Ejemplo de una gráfica, u y v son adyacentes

Definición 14. Sea v un vértice de $G = (V, E)$ el **grado** de v es el número de vértices adyacentes a él y se denota por $deg(v)$. Ver figura 2.13.

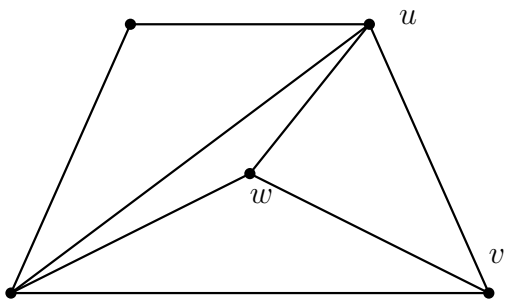


Figura 2.13: En este ejemplo $deg(u) = 4$, $deg(v) = 3$ y $deg(w) = 3$

Definición 15. Un **camino** de v_1 a v_k en una gráfica $G = (V, E)$ es una secuencia de vértices distintos $v_1, v_2, v_3, \cdots, v_k$, con $k > 1$, tal que cada pareja $\{v_i, v_{i+1}\} \in E$ con $1 \leq i < k$. Ver figura 2.14.

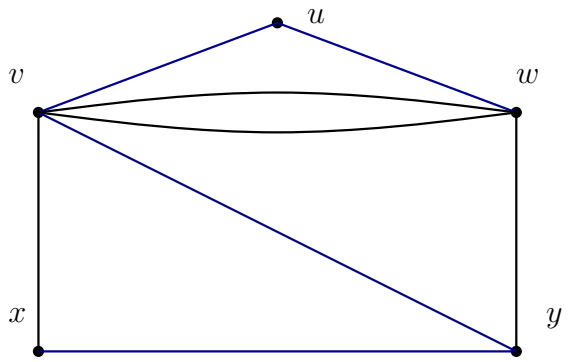


Figura 2.14: La secuencia de vértices x, y, v, u, w, muestra un camino de x a w

Definición 16. Un **ciclo** de $G = (V, E)$ es un camino tal que la secuencia de vértices comienza y termina en el mismo vértice, $v_1, v_2, v_3, \cdots, v_k, v_1$. Ver figura 2.15.

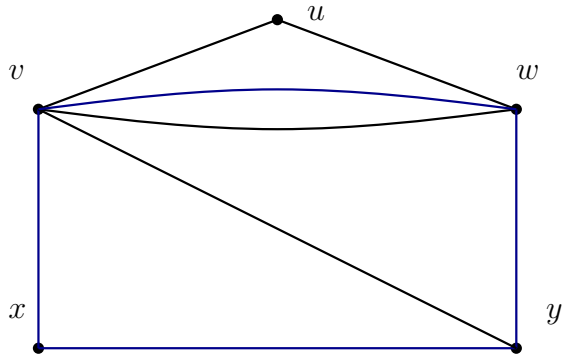
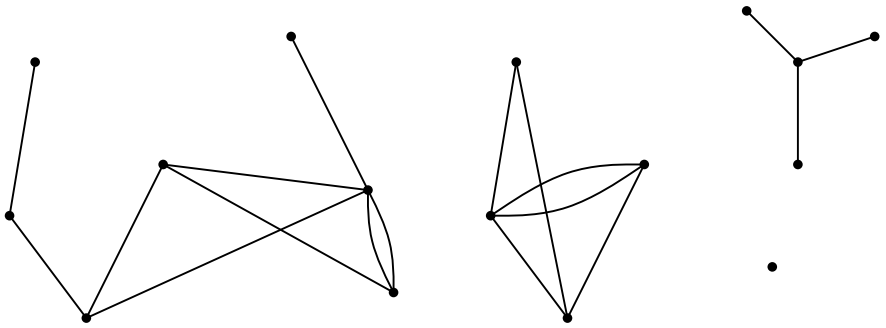


Figura 2.15: La secuencia de vértices x, v, w, y, x, muestra un ciclo dentro de la gráfica

Definición 17. Diremos que una gráfica $G = (V, E)$ es **conexa** si para cualquier par de vértices $u, v \in V$, existe un camino de u a v, $u = v_1, v_2, v_3, \cdots, v_k = v$. Ver figura 2.16.



(a) (b)

Figura 2.16: (a) Ejemplo de una gráfica conexa, (b) Ejemplo de una gráfica no conexa con tres componentes

Definición 18. Diremos que una gráfica $G = (V, E)$ es un **árbol** si G es conexa y no contiene ciclos. Ver figura 2.17.

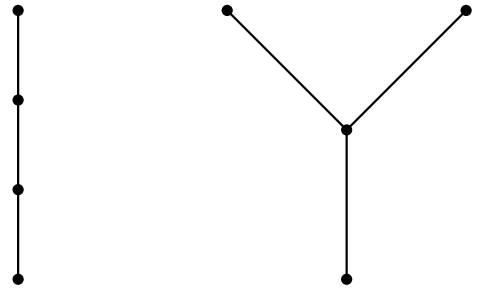


Figura 2.17: Los árboles de cuatro vértices

Definición 19. Una gráfica $G = (V, E)$ es **incrustable en el plano** o **plana** si puede ser dibujada en el plano de tal forma que sus aristas se intersectan solamente en sus extremos. Ver figura 2.18.

Definición 20. Las **caras interiores** de una gráfica plana G son las regiones del plano acotadas por las aristas de G. La región no acotada es llamada la **cara exterior**.

Definición 21. El **grado** de una cara es el número de aristas que forman la frontera de ésta.

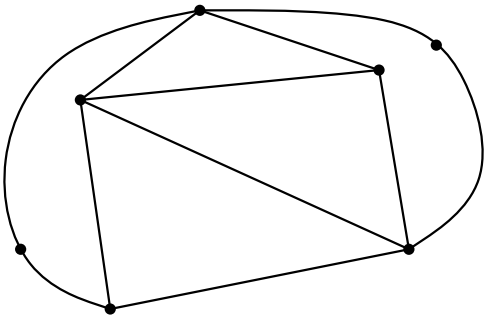


Figura 2.18: Ejemplo de una gráfica plana

Definición 22. La **gráfica dual** $G^* = (V^*, E^*)$ de una gráfica $G = (V, E)$ se define como sigue:
Para cada cara f de G, V^* contiene un vértice v_f. Para cada arista $e \in E$ se tiene una arista $e^* = (v_{f_1}, v_{f_2}) \in E^*$ donde f_1 y f_2 son dos caras de G, con e en su frontera común. Ver figura 2.19.

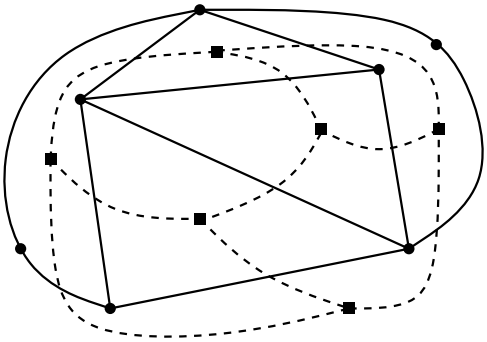


Figura 2.19: Ejemplo de una gráfica plana y su gráfica dual

Definición 23. Sea $G = (V, E)$ una gráfica. Una **coloración** de los vértices de G es una asignación de colores a sus vértices de tal manera que dos vértices adyacentes reciben colores diferentes. Una *k-coloración* es una coloración que utiliza k colores distintos. Ver figura 2.20.

2.7. Notación asintótica

El orden de crecimiento del tiempo de ejecución de un algoritmo nos da una caracterización simple de la eficiencia del mismo, esta caracterización se puede utilizar para comparar el desempeño relativo de un algoritmo con respecto a otros algoritmos alternativos.

En esta sección se describe la notación que se utiliza en este trabajo para medir la eficiencia asintótica de los algoritmos definidos a lo largo del mismo. Las notaciones

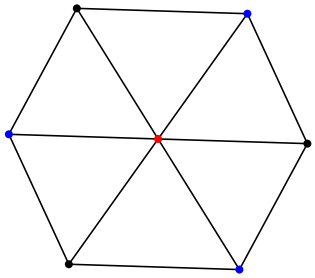


Figura 2.20: Ejemplo de una 3-coloración

que usaremos para describir el tiempo de ejecución de un algoritmo están definidas en términos de funciones con dominio en el conjunto de los números naturales $\mathbb{N}$.

Las siguientes definiciones se pueden consultar ampliamente en [14].

Definición 24. La **notación de O grande** es utilizada para representar una **cota superior** de forma asintótica del crecimiento de una función para una entrada suficientemente grande. Ver figura 2.21.
Para una función $g(n)$ denotamos por $O(g(n))$ al conjunto de funciones:

$$O(g(n)) = \{\, f(n) : \text{existen } c, n_0 \in \mathbb{N} \text{ tales que } 0 \leq f(n) \leq cg(n) \text{ para toda } n \geq n_0 \}$$

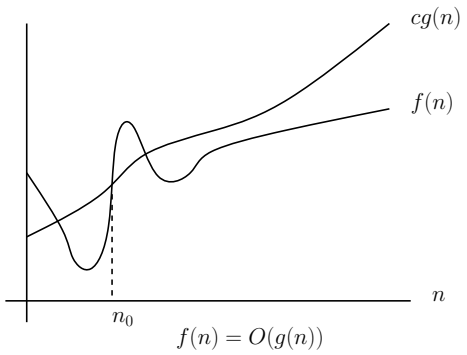


Figura 2.21: Ilustración de la definición 23.

Definición 25. La notación Ω denota una **cota inferior** asintótica. Ver figura 2.22. Dada una función $g(n)$ denotamos por $\Omega(g(n))$ al conjunto de funciones:

$$\Omega(g(n)) = \{\, f(n) : \text{existen } c, n_0 \in \mathbb{N} \text{ tales que } 0 \leq cg(n) \leq f(n) \text{ para toda } n \geq n_0 \}$$

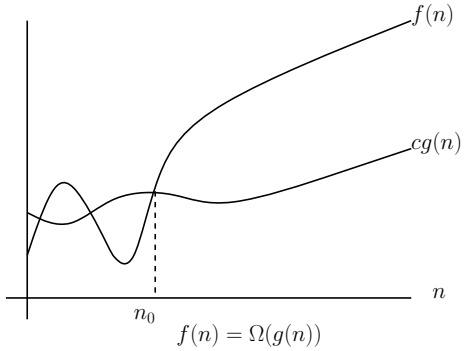


Figura 2.22: Ilustración de la definición 24.

Definición 26. La notación Θ denota la **cota asintótica superior e inferior**, es decir denota la **cota justa**. Ver figura 2.23. Para una función $g(n)$ denotamos por $\Theta(g(n))$ al conjunto de funciones:

$$\Theta(g(n)) = \left\{ \begin{array}{l} f(n) : \text{existen } c_1, c_2, n_0 \in \mathbb{N} \text{ tales que } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para toda} \\ n \geq n_0 \end{array} \right\}$$

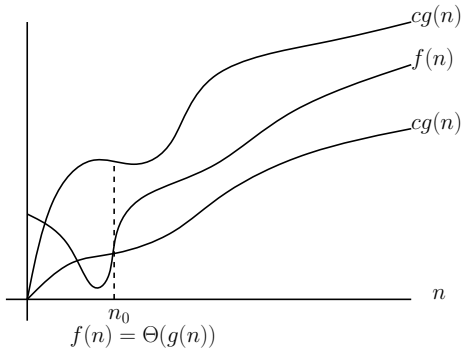


Figura 2.23: Ilustración de la definición 25.

Capítulo 3

Algoritmos para la construcción de polígonos

Las propiedades de cada uno de los polígonos que estudiaremos en este capítulo se han descrito en el capítulo anterior. La elección de estos tipos de polígono se dio tomando en cuenta los diversos problemas que surgen al estudiar la iluminación en cada uno de ellos. En la aplicación estos polígonos son construidos de manera guiada por el usuario, es decir, una vez elegido un tipo, la aplicación guía al usuario a dibujarlo de forma correcta. Esto lo hace verificando que las propiedades del polígono se conserven al momento de insertar un vértice nuevo, de esta forma no se permite agregar vértices que violen las propiedades del tipo de polígono que se eligió.

En las siguientes secciones se muestran con detalle los algoritmos que se diseñaron para la construcción de los polígonos utilizados en la aplicación.

3.1. Polígonos simples

La principal característica de los polígonos simples es la de no permitir intersecciones entre sus aristas, como ya se mencionó antes, se desea un algoritmo que de cierta forma guíe la construcción del polígono.

3.1.1. Construcción

Comenzaremos por describir un algoritmo que verifique el cumplimiento de dicha característica cada vez que tratamos de insertar un vértice nuevo en un polígono simple P, el algoritmo debe recibir como entrada un vértice v y regresar un valor de verdad, **True** o **False**, como salida.

Sea $a = (v_{i-1}, v_i)$ la arista formada por los vértices v_{i-1} y v_i, donde v_{i-1} es el último vértice insertado en P y v_i el nuevo vértice que se quiere agregar. El algoritmo debe verificar que a no intersecta al resto de las aristas de P, denotadas por $A(P)$.

Algoritmo 1: *esSimple(v_i)*

Data: v_i, el vértice que deseamos insertar a nuestro polígono simple P. Denotamos las aristas de P como $A(P)$.

Result: $\begin{cases} \textbf{False}, & \text{si } a_i = (v_{i-1}, v_i) \text{ intersecta } A(P) \\ \textbf{True}, & \text{en otro caso} \end{cases}$

```
1 begin
2     if A(P) es vacío then
3         return True
4     for a_j ∈ A(P) do
5         if a_j ∩ a_i = (v_{i-1}, v_i) then
6             return False
7     return True
```

El algoritmo *esSimple* verifica la intersección de una arista a_i con un conjunto de aristas $A(P)$, pero solo se hace referencia a la operación de intersección entre aristas. Esta operación es una de las más comunes y utilizadas en la geometría computacional y existen varios métodos para efectuar dicha operación, en este trabajo se utilizó el algoritmo presentado en [30]. A grandes rasgos el algoritmo es: dadas dos aristas a y b, verificar si los vértices d a se encuentran en lados opuestos con respecto a la línea que contiene a b y viceversa.

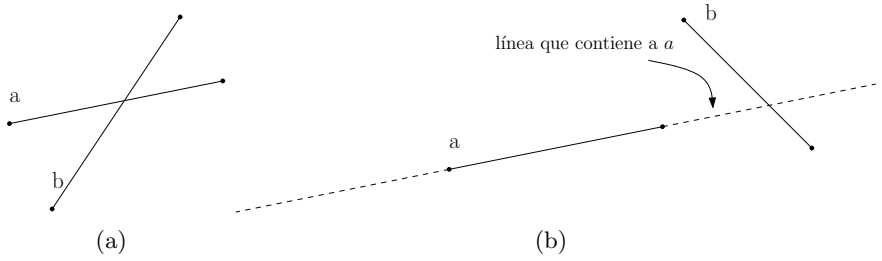


Figura 3.1: (a) Intersección entre a y b (b) No hay intersección

Sea v_i el vértice que se desea insertar en P y $V(P)$ los vértices de P, utilizando *esSimple* podemos definir un algoritmo para insertar vértices en un polígono simple asegurándonos que el vértice nuevo preserve las propiedades de polígono simple, a este algoritmo lo llamaremos *insertaSimple*. Ver algoritmo 2.

InsertaSimple recibe como entrada a v_i y regresa como salida un valor de verdad, **True** si v_i se inserto con éxito, o **False** en otro caso. Notemos que al inicio de la construcción el conjunto de vértices y el conjunto de aristas son vacíos, por lo que la verificación de intersección de aristas no tiene sentido hasta que tengan al menos dos aristas.

Esta operación se realiza hasta *cerrar* el polígono, es decir, unir el primer vértice insertado con el último. Al *cerrar* el polígono se tiene que verificar que la arista que cierra también

Algoritmo 2: *insertaSimple(v_i)*

Data: v_i, el vértice que deseamos insertar a nuestro polígono simple P. Denotamos los vértices de P como $V(P)$.

Result: $\begin{cases} \textbf{True}, & \text{si } v_i \text{ se insertó en } P \\ \textbf{False}, & \text{en otro caso} \end{cases}$

```
1  begin
2      if A(P) es vacío then
3          if |V(P)| = 0 then
4              V(P) = V(P) ∪ v_i
5              return True
6          else
7              V(P) = V(P) ∪ v_i
8              a = (v_{i-1}, v_i)
9              A(P) = A(P) ∪ a
10             return True
11     if esSimple(v_i) then
12         V(P) = V(P) ∪ v_i
13         a = (v_{i-1}, v_i)
14         A(P) = A(P) ∪ a
15         return True
16     else
17         return False
```

cumpla con las propiedades de un polígono simple, si no cumple no se permitirá cerrar el polígono. Notemos también que no se puede construir un polígono con menos de 3 vértices diferentes, por lo que antes de *cerrar* se debe verificar que la cantidad de vértices sea mayor o igual a 3 y que éstos sean diferentes.

Realizando estas verificaciones y usando *insertaSimple* se puede definir el algoritmo interactivo *poligonoSimple*, ver algoritmo 3. El algoritmo recibe como entrada un conjunto de vértices elegidos en tiempo real por el usuario. El obtener un par de coordenadas (x, y) para formar un vértice de forma interactiva depende del dispositivo de entrada que se utilice, por lo que esta acción se representa aquí y en las siguientes secciones de este capítulo por medio de la función *obtenVertice*. No se considera necesario definir formalmente dicha función, basta con entender que al ejecutarla se obtiene un vértice, el cual fue elegido por el usuario.

Al terminar *poligonoSimple* regresa como salida un polígono simple formado con los vértices válidos que eligió el usuario.

Algoritmo 3: *poligonoSimple()*

```
   Data:
   Result: P, polígono después de insertar al menos 3 vértices
1  begin
2      P = ∅                                                  // polígono vacío
3      while True do
4          v = obtenVertice()
5          if v ∈ V(P) then
6              if |V(P)| < 3 then
7                  continue                          // comienza una nueva iteración
8              if v = v_0 and insertaSimple(v) then
9                  return P
10         else
11             insertaSimple(v)
```

3.1.2. Complejidad

Sea P, el polígono resultante de usar *poligonoSimple*, con $|V(P)| = n$, entonces $|A(P)| = n$. Es fácil ver que la complejidad de *esSimple* es de $O(n)$, pues este algoritmo en las líneas 4-6 recorre todas las aristas de P para verificar que la nueva arista no intersecta a ninguna de las aristas ya existentes. Notemos que la operación de intersección entre dos aristas toma tiempo $O(1)$, entonces cada iteración de las líneas 4-6 en *esSimple* realiza una operación de intersección, en total se verifican n intersecciones.

Ahora analizaremos la complejidad de *insertaSimple*, notemos que este algoritmo realiza una verificación antes de insertar el vértice nuevo en P, esta verificación se realiza en la línea 11, usando solo una vez la función *esSimple*, entonces la complejidad está dada por la complejidad de *esSimple* más la complejidad de insertar un vértice nuevo en P, esto es: $n + c$, con c una constante, lo cual implica que la complejidad es de $O(n)$.

Consideremos a m, la cantidad de veces que se llama a la función *obtenVertice*, es decir la cantidad de vértices que el usuario trató de insertar en P. Dado que solo se agregan a P los vértices que satisfacen las verificaciones descritas en los algoritmos anteriores, es fácil ver que $m \geq n$. Para acotar la complejidad de *polígonoSimple*, analicemos esto en dos casos:

Caso 1: $m > n$, sea $k = m - n$, la cantidad de vértices que no se agregaron a P, para determinar que estos vértices violan las propiedades del *polígono simple*, tuvieron que ser sometidos a todas las verificaciones que realiza el algoritmo en las líneas 8 y 11, dichas verificaciones las realiza el algoritmo *insertaSimple*, entonces la complejidad de discriminar estos k vértices es del orden de $O(k \times n)$.
Ahora veamos que pasa con los $m - k$ vértices faltantes, estos vértices pasaron todas

las verificaciones y además fueron agregados como vértices de P, análogamente al caso anterior, el tiempo que toma realizar las verificaciones y agregarlos a P es del orden de $O((m-k)\times n)$
En total el algoritmo realiza:

$$\begin{aligned}(k\times n)+((m-k)\times n)& \\ &= (k\times n)+(m\times n)-(k\times n) \\ &= m\times n\end{aligned} \tag{3.1}$$

Como $m > n$ podemos decir que $m \times n < m^2$, entonces la complejidad del algoritmo se puede acotar por $O(m^2)$.

Caso 2: $m = n$, en este caso todos los vértices pasaron las verificaciones del algoritmo y se agregaron a P, entonces la complejidad del algoritmo es de $O(m \times n)$, como $m = n$ podemos acotar la complejidad por $O(m^2)$.

Por los casos **1** y **2**, podemos concluir que la complejidad de *poligonoSimple* es $O(m^2)$. Para poner la complejidad en términos de $|V(P)| = n$ hagamos la siguiente aclaración, se espera que todos los vértices que el usuario elige se agreguen a P, es decir se espera que $m = n$, así la complejidad de *poligonoSimple* es $O(n^2)$.

El análisis para este último algoritmo es prácticamente el mismo para los algoritmos equivalentes en las siguientes secciones, para las cuales se realizará un análisis más compacto, bajo el supuesto de que $m = n$.

3.2. Polígonos monótonos

En el capítulo anterior se definen los polígonos *monótonos*, cuya característica principal es: dada una recta o una dirección, el polígono siempre es creciente con respecto a ésta, el algoritmo descrito en esta sección solo contempla la construcción de polígonos *x-monótonos*.

3.2.1. Construcción

Siguiendo la estructura del algoritmo anterior, primero se definirá una función para determinar si el nuevo vértice que se desea insertar en un polígono monótono P preserva las características de su tipo. Dicha prueba de monotonía depende de la configuración de la porción del polígono en construcción que se tiene al momento de insertar un nuevo vértice. Las configuraciones se pueden diferenciar por la cantidad de *regresos* o cambios de dirección que permite este tipo de polígono. Puesto que sólo se trabaja con polígonos *x-monótonos*, podemos decir que las dos únicas direcciones que puede tener P son: *izquierda* y *derecha*.

Definición 27. Sea P un polígono monótono. Diremos que un regreso es invertir la dirección que define la monotonía de P. Ver figura 3.2.

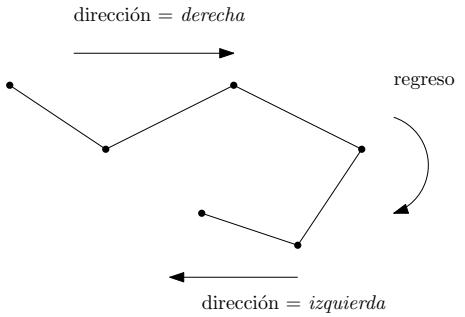


Figura 3.2: Direcciones de un polígono *x-monótono*

Es fácil ver que un polígono *x-monótono* tiene exactamente 2 *regresos*, si tuviera menos el polígono no ha sido cerrado, si tuviera más entonces el polígono ya no sería monótono. Ver figura 3.3

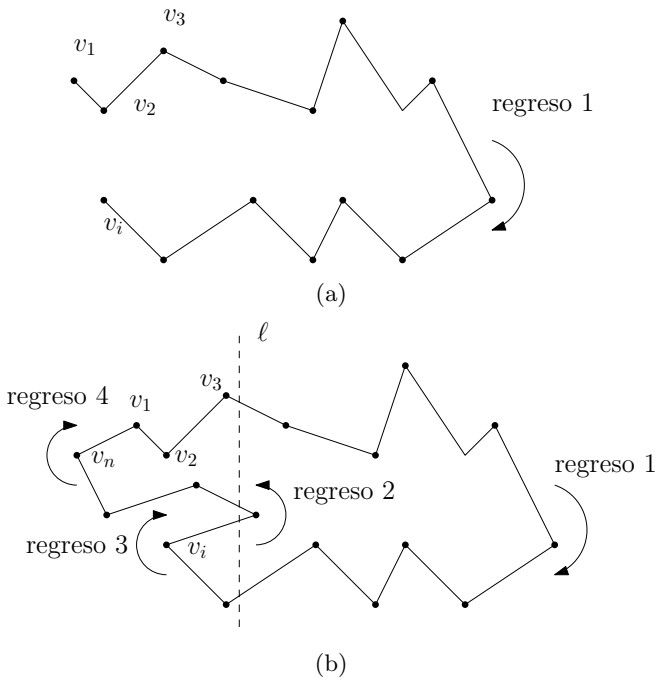


Figura 3.3: (a) 1 regreso, no está cerrado (b) 4 regresos, la recta ℓ corta el polígono en más de 2 puntos, el polígono no es monótono

Ahora, es necesario tener una función que verifique si v genera un regreso al agregarlo a P. Esta función, la cual llamaré *regreso* (ver algoritmo 4), resulta sencilla, sea v el vértice que queremos agregar, u el último vértice agregado y d la dirección de la última

arista agregada, la función verifica si v está más a la *izquierda* o más a la *derecha* de u, dependiendo del valor de d el resultado es **True** o **False**.

Algoritmo 4: *regreso*

Data: v, el vértice que deseamos insertar; u, el último vértice agregado; d, dirección de la última arista agregada

Result: **True**, si v genera un retroceso; **False**, en otro caso

```
1 begin
2     if A(P) = ∅ then
3         return False
4     if d = derecha and v.x < u.x then
5         return True
6     if d = izquierda and v.x > u.x then
7         return True
8     return False                                  // en otro caso
```

También necesitamos una función *giro*, dados un punto v y un segmento de recta dirigido s, esta función determina de qué lado de la recta que contiene a s se encuentra ubicado v. El algoritmo usado se puede encontrar en [30].

El algoritmo consiste en calcular el área del triángulo formado por los vértices de s y el punto v a partir de las coordenadas de sus vértices. A partir del valor del área podemos determinar el lado en el que se encuentra v con respecto a s:

- Si el área es positiva (área > 0) entonces v se encuentra a la *izquierda* de s.
- Si el área es negativa (área < 0) entonces v se encuentra a la *derecha* de s.
- Si el área es cero (área $= 0$) entonces v y s son colineales.

En el siguiente apartado se explica a detalle la forma de calcular el área del triángulo.

Área del triángulo

Gracias al álgebra lineal sabemos que la magnitud del producto cruz de dos vectores es el área del paralelogramo que éstos determinan. Sean A y B dos vectores, entonces $|A \times B|$ es el área del paralelogramo con lados A y B, ver figura 3.4.

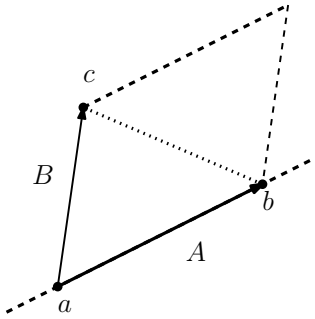


Figura 3.4: Paralelogramo producto cruz

Un triángulo puede verse como la mitad del paralelogramo, esto nos permite utilizar el producto cruz para determinar el área de un triángulo a partir de las coordenadas de los vértices que lo forman. Definamos $A = b - a$ y $B = c - a$, entonces el área del triángulo es igual a la mitad de $|A \times B|$. Este producto cruz se puede calcular por el siguiente determinante, donde i, j, k son los vectores unitarios en las direcciones de x, y y z respectivamente.

$$\begin{vmatrix} i & j & k \\ A_0 & A_1 & A_2 \\ B_0 & B_1 & B_2 \end{vmatrix} = (A_1B_2 - A_2B_1)i + (A_2B_0 - A_0B_2)j + (A_0B_1 - A_1B_0)k \tag{3.2}$$

Para vectores en $\mathbb{R}^2$ tenemos que: $A_2 = B_2 = 0$. Lo que reduce el cálculo a $(A_0B_1 - A_1B_0)k$. El producto cruz da como resultado un vector perpendicular al plano que contiene al triángulo. Por lo que el área está dada por:

$$Area(T) = \frac{1}{2}(A_0B_1 - A_1B_0) \tag{3.3}$$

Sustituyendo $A = b - a$ y $B = c - a$ tenemos:

$$\begin{aligned} 2 \times Area(T) &= (b_0 - a_0)(c_1 - a_1) - (c_0 - a_0)(b_1 - a_1) \\ &= a_0b_1 - a_1b_0 + a_1c_0 - a_0c_1 + b_0c_1 - c_0b_1 = \begin{vmatrix} a_0 & a_1 & 1 \\ b_0 & b_1 & 1 \\ c_0 & c_1 & 1 \end{vmatrix} \end{aligned} \tag{3.4}$$

Regresando a la construcción de polígonos monótonos. Antes de definir los casos que se deben contemplar según la cantidad de regresos, necesitamos otra función que nos ayude a comparar la posición de dos vértices con respecto a alguna dirección, es decir, dados dos vértices v_1, v_2 y una dirección, queremos saber qué vértice está más *cargado* hacia esta dirección. Como ya se mencionó antes (ver sección 3.2), en esta sección solo consideramos

dos direcciones (*izquierda* y *derecha*). Por lo que diremos que: si la dirección es *derecha* v_1 *domina* a v_2 si $v_1.x \geq v_2.x$, si la dirección es *izquierda* v_1 *domina* a v_2 si $v_1.x \leq v_2.x$. Ver algoritmo 5.

Algoritmo 5: *domina*

Data: $\begin{cases} v_1, & \text{vértice} \\ v_2, & \text{vértice} \\ d, & \text{dirección} \end{cases}$

Result: $\begin{cases} \textbf{True}, & \text{si } v_1 \text{ domina a } v_2 \\ \textbf{False}, & \text{en otro caso} \end{cases}$

```
1 begin
2     if d = derecha then
3         return v1.x ≥ v2.x
4     else
5         return v1.x ≤ v2.x
```

Con base en el número de regresos que tiene la construcción parcial de P, las restricciones para agregar un vértice nuevo son diferentes para los siguientes casos:

regresos = 0: En este caso la única restricción es que las aristas no se intersecten entre sí, es decir, que cumplan con la propiedad de polígono simple.

regresos = 1: Las restricciones en este caso son: primero, solo se admite otro regreso si v_i *domina* v_0 y v *domina* v_0; segundo, si v genera un regreso válido, sea g_0 el giro que realiza el primer regreso y g_1 el giro realizado por v, si $g_0 = derecha$ entonces g_1 debe ser igual a *derecha*, análogamente, si $g_0 = izquierda$ entonces g_1 debe ser igual a *izquierda*. Por último, no se debe permitir que las aristas se intersecten entre sí. Ver figura 3.5

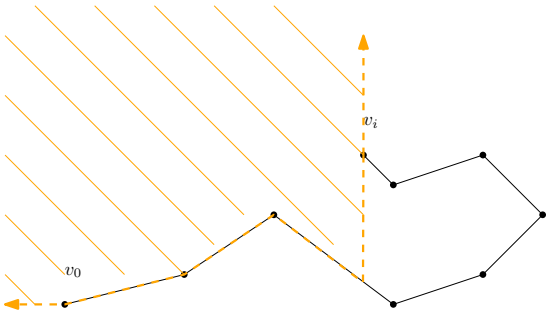


Figura 3.5: Área donde se puede agregar a v

regresos = 2: Para este caso el espacio de elección queda acotado por v_0 y v_i, para que P conserve la propiedades de un polígono monótono, se debe cumplir que v no genere algún regreso, que v_0 *domine* v y que las aristas no se intersecten entre sí. Ver figura 3.6.

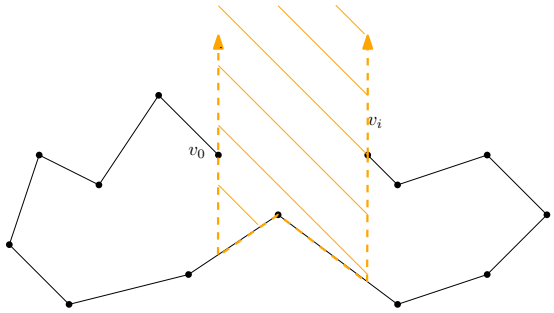


Figura 3.6: Área donde se puede agregar a v

Una vez definidos estos casos, se define el algoritmo *esMonotono* (ver algoritmo 6), este algoritmo recibe como entrada a v y como salida un valor de verdad, **True**, si al insertar v en P, P conserva las propiedades de un polígono monótono, o **False** en otro caso. Sin pérdida de generalidad, consideremos que la dirección inicial de P es $d = derecha$, el caso para $d = izquierda$ es análogo.

Cabe aclarar que este algoritmo sólo verifica la conservación de las propiedades de P, en ningún paso se realiza la operación de insertar, esta operación se realizará solo después de que v pase la verificación.

Algoritmo 6: *esMonotono*

Data: v, el vértice que deseamos insertar a nuestro polígono monótono P.
Result: { **True**, si se verifica que al insertar v se conserva un polígono monótono; **False**, en otro caso

```
1  begin
2      if |V(P)| = ∅ then
3          inicio = v
4          return True
5      if r = 0 then
                                                  // r = cantidad de regresos
6          if regreso(v) then
7              g = giro(a_i, v)
                                          // a_i = última arista agregada a P
8              r++
9          return True
10     if r = 1 then
11         if regreso(v) then
12             if giro(a_i, v) ≠ g then
13                 return False
14             if domina(inicio, v) then
15                 return False
16             r++
17             return True
18         else
19             return True
20     if r = 2 then
21         if regreso(v) then
22             return False
23         if domina(v, inicio) then
24             return False
25         return True
```

El siguiente algoritmo agrega un vértice v_i a P siempre que v_i cumpla con las restricciones del polígono *x-monótono*, a este algoritmo lo llamaremos *insertaMonotono*. Ver algoritmo 7.

Algoritmo 7: *insertaMonotono(v_i)*

Data: v_i, el vértice que deseamos insertar a nuestro polígono monótono P.
Denotamos los vértices de P como $V(P)$.

Result: $\begin{cases} \textbf{True}, & \text{si } v_i \text{ se inserta en } P \\ \textbf{False}, & \text{en otro caso} \end{cases}$

```
1  begin
2      if A(P) = ∅ then
3          if |V(P)| = 0 then
4              V(P) = V(P) ∪ v_i
5              return True
6          else
7              V(P) = V(P) ∪ v_i
8              a = (v_{i-1}, v_i)
9              A(P) = A(P) ∪ a
10             return True
11     if esMonotono(v_i) and esSimple(v_i) then
12         V(P) = V(P) ∪ v_i
13         a = (v_{i-1}, v_i)
14         A(P) = A(P) ∪ a
15         return True
16     else
17         return False
```

Por último se define el algoritmo *poligonoMonotono* (ver algoritmo 8), el cual construye un polígono *x-monótono* leyendo interactivamente los vértices de entrada, este proceso es el mismo que se define en la sección anterior, por lo que usaremos para esto la función *obtenVertice()*.

Recordemos que, apegándonos a nuestra definición de polígono, se deben tener al menos 3 vértices diferentes para formar un polígono. Además de que la forma de terminar la construcción de nuestro polígono P, es unir de forma válida el primer y el último vértices de P.

Algoritmo 8: *poligonoMonotono*

```
   Data:
   Result: P, polígono monótono después de insertar al menos 3 vértices
1  begin
2      P = ∅                                                  // polígono vacío
3      while True do
4          v = obtenVertice()
5          if v ∈ V(P) then
6              if |V(P)| < 3 then
7                  continue                          // comienza una nueva iteración
8              if v = v_inicio and insertaMonotono(v) then
9                  return P
10         else
11             insertaMonotono(v)
```

3.2.2. Complejidad

Sea $n = |V(P)|$, analizaremos la complejidad de construir un polígono monótono. Analizar la complejidad de *regreso* resulta sencillo, notemos que en las líneas 2, 4 y 6, se realizan 3 verificaciones, una por cada línea, cada verificación toma tiempo constante, por lo que la complejidad de *regreso* es de $O(1)$. De forma similar *domina* realiza 2 comparaciones, cada una de tiempo constante, entonces la complejidad de *domina* es de $O(1)$. La complejidad de la función *giro*, está dada por el cálculo de un determinante de 2×2, lo cual es claro que toma tiempo $O(1)$.

El algoritmo *esMonotono* es un poco más complicado, veamos que en las líneas 5, 10 y 20 se tienen 3 casos, uno en cada línea:
r = 0, en el peor de los casos se realizan 3 operaciones, cada una de tiempo constante.
r = 1, el peor caso es que v genere un regreso válido, pues se tienen que realizar las verificaciones de las líneas 11, 12 y 14, además de incrementar la variable r, en total se realizan 4 operaciones de orden constante.
r = 2, en este caso se realizan a lo más 2 verificaciones, una en la línea 21 y otra en la línea 23, cada una de tiempo constante.

Notemos que cada caso toma tiempo $O(1)$ y que cada caso es independiente, es decir, un vértice no puede caer en dos distintos casos durante la ejecución del algoritmo, así que la complejidad de *esMonotono* está dada por la comparación de la línea 2 más la complejidad del caso en el que cae la ejecución, puesto que cada caso toma tiempo $O(1)$, podemos decir que la complejidad total de dicho algoritmo es de $O(1)$.

insertaMonotono se puede analizar en dos casos, el primero es cuando $A(P) = \varnothing$, en

este caso no se tiene ninguna arista, ni alguna dirección para realizar las verificaciones correspondientes, la complejidad de este caso está dada por la operación de agregar un vértice nuevo a la lista $V(P)$, esto es de $O(1)$.
El segundo caso es cuando $A(P) \neq \varnothing$, en este caso se realiza una llamada en la línea 11 a *esMonotono* y a *esSimple*, lo cual sabemos toma tiempo $O(1)$ y $O(n)$ respectivamente, luego en las líneas 12-14 se agrega el vértice nuevo y la arista recién formada, lo que cuesta tiempo $O(1)$. En el peor de los casos se deben realizar las verificaciones del segundo caso, lo que implica que la complejidad de *insertaMonotono* es de tiempo $O(n)$.
Finalmente la complejidad de *poligonoMonotono* está dada por la cantidad de veces que se ejecutan las verificaciones de la línea 8 multiplicado por le complejidad de *insertaMonotono*, esto es $n \times n$, entonces la complejidad de este algoritmo es de $O(n^2)$.

3.3. Polígonos ortogonales

Los polígonos ortogonales han sido objeto de particular interés en el estudio de problemas de vigilancia e iluminación debido quizá a la eficiencia y elegancia de los resultados obtenidos sobre estos polígonos. Ejemplos de problemas y resultados obtenidos sobre el estudio de estos polígonos se pueden revisar detalladamente en [29].

3.3.1. Construcción

La característica principal de un polígono ortogonal P es que todos sus lados son paralelos a los ejes coordenados. En una aplicación interactiva donde el usuario elige los vértices de P seleccionando gráficamente un punto sobre un *lienzo*, resulta difícil elegir dos puntos que tengan el mismo valor en la coordenada x o en la coordenada y, por lo que se diseñó un algoritmo para asistir al usuario en esta tarea. Ver algoritmo 9.
El algoritmo es sencillo, primero toma como origen el último vértice insertado u y se verifica en que cuadrante de este sistema coordenado se encuentra v, el vértice que se desea insertar. Una vez determinado esto, se toma la recta ℓ que bisecta el ángulo formado por las rectas del cuadrante y que pasa por el origen, luego se determina de qué lado de ℓ se encuentra v, esto nos dirá a cual de los ejes se encuentra más cerca de v, por último movemos v sobre el eje determinado anteriormente proyectándolo sobre el mismo. Esto nos garantiza que el segmento de línea $\overline{uv}$ sea paralelo a alguno de los ejes coordenados.
Notemos que en la línea 5 implícitamente se determina el cuadrante que contiene a v, esta operación no se explica a detalle pues es sencillo realizar dicha verificación. Al conocer el cuadrante que contiene a v, se tienen que analizar cuatro casos, uno por cada cuadrante. Si tomamos a ℓ como una recta dirigida podemos ver que los casos de los cuadrantes I y III son simétricos, ver figura 3.7, análogamente para los cuadrantes II y IV. Agrupando estos casos se reducen a dos.
Notemos también que al momento de decidir de qué lado de ℓ se encuentra v no se ha contemplado el caso donde v está sobre ℓ, en ese caso se puede definir a qué eje se proyectará v, en *calculaPuntoOrtogonal* se trata como si v estuviese del lado derecho de ℓ.
En el algoritmo anterior se llama a la función $giro(s, v)$, ya descrita en la sección 3.1, esta función determina de qué lado de la recta que contiene a s se encuentra ubicado v. El

Algoritmo 9: *calculaPuntoOrtogonal*

```
Data: v, el vértice que elige el usuario.
Result: v_o, el vértice ortogonal
1  begin
2      if V(P) = ∅ then
3          return v,
4      u = vértice anterior;
5      cuadrante = cuadrante que contiene a v ;
6      if cuadrante = I o cuadrante = III then
7          ℓ = recta que bisecta el cuadrante por el origen;
8          if giro(ℓ, v) = derecha o giro(ℓ, v) = colineal then
9              v_o = (v.x, u.y);
10         if giro(ℓ, v) = izquierda then
11             v_o = (u.x, v.y);
12     if cuadrante = II o cuadrante = IV then
13         ℓ = recta que bisecta el cuadrante por el origen;
14         if giro(ℓ, v) = derecha o giro(ℓ, v) = colineal then
15             v_o = (u.x, v.y);
16         if giro(ℓ, v) = izquierda then
17             v_o = (v.x, u.y);
18     return v_o
```

algoritmo usado se puede encontrar en [30].

Ahora que ya podemos garantizar la construcción de aristas paralelas a los ejes coordenados se describirá la función *insertaOrtogonal*, ver algoritmo 10, esta función toma como entrada un vértice v, verifica que se forme una arista ortogonal y si cumple con las propiedades de polígono ortogonal lo inserta en el polígono P. Utilizando *calculaPuntoOrtogonal* se construyen las aristas ortogonales, la propiedad que falta por verificar es que las aristas de P no se intersecten entre sí, para realizar dicha verificación se ha utilizado la función *esSimple()* descrita en la sección 3.1.

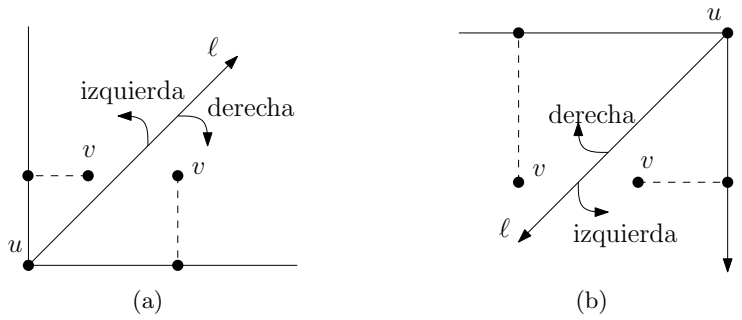


Figura 3.7: (a) v está en el cuadrante I (b) v está en el cuadrante III

Algoritmo 10: *insertaOrtogonal(v_i)*

Data: v_i, el vértice que deseamos insertar a nuestro polígono ortogonal P.
Result: { **True**, si v_i se inserta en P ; **False**, en otro caso

```
1  begin
2      if A(P) = ∅ then
3          if |V(P)| = 0 then
4              V(P) = V(P) ∪ v_i
5              return True
6          else
7              V(P) = V(P) ∪ v_i
8              a = (v_{i-1}, v_i)
9              A(P) = A(P) ∪ a
10             return True
11     if esSimple(v_i) then
12         V(P) = V(P) ∪ v_i
13         a = (v_{i-1}, v_i)
14         A(P) = A(P) ∪ a
15         return True
16     else
17         return False
```

Por último se define el algoritmo *poligonoOrtogonal*, ver algoritmo 11. Similar a los algoritmos de las secciones anteriores, este algoritmo recibe como entrada un conjunto de vértices elegidos de forma interactiva por el usuario y regresa como salida a P, el polígono ortogonal construido con dichos vértices.

Algoritmo 11: *poligonoOrtogonal*

```
   Data:
   Result: P, polígono ortogonal después de insertar al menos 3 vértices
1  begin
2      P = ∅                                                  // polígono vacío
3      while True do
4          v = obtenVertice()
5          v_o = calculaPuntoOrtogonal(v)
6          if v_o ∈ V(P) then
7              if |V(P)| < 3 then
8                  continue                          // comienza una nueva iteración
9              if v_o = v_inicio and insertaOrtogonal(v_o) then
10                 return P
11         else
12             insertaOrtogonal(v_o)
```

3.3.2. Complejidad

Consideremos $|V(P)| = n$, analicemos la complejidad de *calculaPuntoOrtogonal*, primero en la línea 5 se realiza implícitamente una operación para determinar el cuadrante que contiene a v, luego se pueden apreciar dos casos, demarcados por las líneas 6 y 12, en cada línea se realizan dos comparaciones y en cada caso se realizan a lo más tres llamadas a la función *giro*, la cual sabemos cuesta tiempo $O(1)$ cada una, por lo que la complejidad de *calculaPuntoOrtogonal* es de $O(1)$.

La complejidad de *insertaOrtogonal* está dada principalmente por la llamada a la función *esSimple* en la línea 11, la cual toma tiempo $O(n)$, por lo que la complejidad de *insertaSimple* es de $O(n)$.

Finalmente *poligonoOrtogonal* toma tiempo igual a la cantidad de vértices agregados multiplicado por la cantidad de llamadas a *insertaOrtogonal* en las líneas 9 y 12, sumado a la cantidad de llamadas a *calculaPuntoOrtogonal* en la línea 5. Entonces *poligonoOrtogonal* toma tiempo de $O(n \times n + n) = O(n^2)$.

3.4. Polígonos ortogonales monótonos

3.4.1. Construcción

Los *polígonos ortogonales monótonos* como su nombre lo sugiere, contienen características de polígonos *ortogonales* y *monótonos* a la vez. Así que construir este tipo de polígonos se vuelve tarea fácil una vez definidas dichas características.

Sea P nuestro polígono ortogonal monótono y v el vértice que deseamos insertar, se define el algoritmo *poligonoOrto-Monotono* (ver algoritmo 12), el cual construye en polígono ortogonal monótono a partir del conjunto de vértices elegidos interactivamente por el usuario. Notemos que se utiliza *insertaMonotono* para garantizar las propiedades de monotonía del polígono y se utiliza *calculaPuntoOrtogonal* para forzar los vértices a formar aristas ortogonales.

Algoritmo 12: *poligonoOrto-Monotono*

```
   Data:
   Result: P, polígono ortogonal monótono después de insertar al menos 3 vértices
 1 begin
 2     P = ∅                                                    // polígono vacío
 3     while True do
 4         v = obtenVertice()
 5         v_o = calculaPuntoOrtogonal(v)
 6         if v_o ∈ V(P) then
 7             if |V(P)| < 3 then
 8                 continue                          // comienza una nueva iteración
 9             if v_o = v_inicio and insertaMonotono(v_o) then
10                 return P
11         else
12             insertaMonotono(v_o)
```

3.4.2. Complejidad

La complejidad de *poligonoOrto-Monotono* es análoga a la complejidad de los algoritmos *poligonoMonotono* y *poligonoOrtognal*, esto es congruente puesto que se utilizan las funciones de estos dos últimos tipos de polígonos para verificar que se cumplan las propiedades de ortogonalidad y monotonía. Por lo que la complejidad de *poligonoOrto-Monotono* es de $O(n^2)$.

Capítulo 4

Visibilidad y galerías de arte

Durante una conferencia en Stanford el año de 1973 *V. Klee* formuló la siguiente pregunta: *¿Cuántos guardias necesitamos para vigilar una galería de arte?* En 1975, *Chvátal* [12] mostró que $\lfloor \frac{n}{3} \rfloor$ guardias son siempre suficientes y a veces necesarios para vigilar cualquier polígono con n vértices. Una galería de arte puede representarse como un polígono y la cantidad de guardias necesarios para ser vigilado depende de la cantidad de vértices que lo forman.

En este capítulo se explica a detalle el concepto de visibilidad en el plano, también se realiza un breve recorrido por la historia del problema de vigilancia de la galería de arte considerando diferentes tipos de polígonos y algunas variantes de éste.

4.1. Visibilidad

El concepto de visibilidad puede explicarse de forma sencilla.

Definición 28. Sean x y y dos puntos en el plano y P un polígono. Diremos que x puede **ver** a y (o y es **visible** a x) si y sólo si el segmento $s = \overline{xy}$ está completamente contenido en P: $s \subseteq P$.

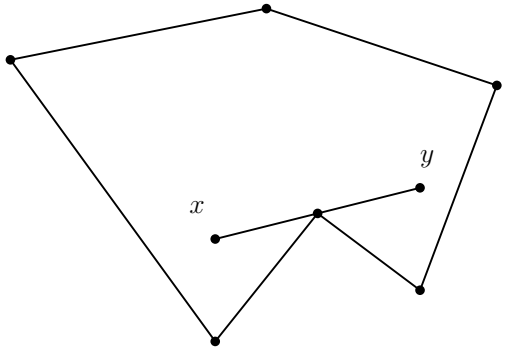


Figura 4.1: Segmento $s = \overline{xy}$, contiene un vértice de P

Notemos que esta definición permite que el segmento s contenga algún vértice de P, como se muestra en la figura 4.1.
Denotemos con δP al conjunto de aristas que delimitan a P, al cual llamaremos la frontera de P. Una definición alternativa que elimina el caso anterior seria:

Definición 29. x es **claramente visible** a y si el segmento $s = \overline{xy} \subseteq P$ y $s \cap \delta P \subseteq \{x, y\}$.

4.2. Triangulación y coloración

La triangulación de polígonos ha sido un tema ampliamente estudiado. El primer algoritmo para triangular un polígono simple fue dado por *M. R. Garey, D. S. Johnson, F. P. Preparata* y *R. E. Tarjan* [23] en el año de 1978, dicho algoritmo toma $O(n \log n)$ tiempo. Esta cota fue mejorada en 1988 por *R. E. Tarjan* y *C. J. Van Wyk* [38], el algoritmo propuesto toma tiempo de $O(n \log \log n)$. Finalmente en 1990, *B. Chazelle* obtuvo el primer algoritmo de tiempo $O(n)$, el resultado de este algoritmo no es fácil de ver y no es objetivo principal, por lo que no se tratará en este trabajo, los detalles pueden consultarse en [10].

Teorema 1. [15] Cualquier polígono simple P con n vértices, admite una triangulación y cualquier triangulación de P consiste exactamente de $n - 2$ triángulos.

Prueba: Probaremos la existencia de la triangulación por inducción sobre n. Cuando $n = 3$ P es un triángulo y el teorema se cumple trivialmente. Si $n > 3$, entonces mostraremos que siempre existe una diagonal que divide a P en dos polígonos, por lo que inductivamente se obtiene el resultado esperado.
Sea v el vértice más a la izquierda de P y sean u y w los vértices adyacentes a v. Si el segmento $s = (u, w)$ es una diagonal, entonces s es la diagonal (ver Definición: 4) que divide a P en dos polígonos. En otro caso, supongamos que s no es una diagonal, entonces el triángulo formado por uvw contiene al menos un vértice de P. Sea v' un vértice de P en el interior del triángulo uvw más alejado de s. Por construcción el segmento $s' = (v, v')$ es la diagonal que buscamos. Ver figura 4.2.

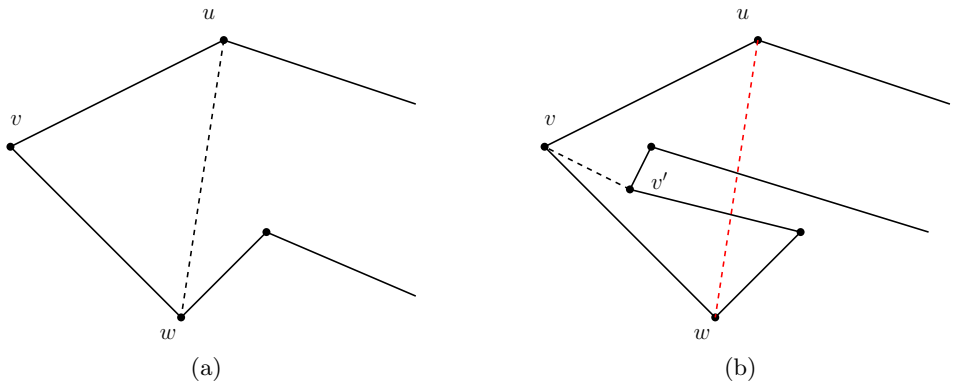


Figura 4.2: (a),(b), Existencia de la diagonal de un polígono

Ahora demostraremos que una triangulación consiste de exactamente $n-2$ triángulos. Sea T una triangulación de P, consideremos una diagonal arbitraria de T. Esta diagonal divide a P en dos polígonos P_1 y P_2, con m_1 y m_2 vértices respectivamente, cada vértice de P se encuentra en P_1 o en P_2, excepto los vértices de la diagonal, que se encuentran en ambos polígonos. Por lo que $m_1 + m_2 = n + 2$. Por inducción cualquier triangulación de P_i consiste de $m_i - 2$ triángulos, lo que implica que:

$$(m_1 - 2) + (m_2 - 2) = n - 2$$

□

El siguiente teorema muestra que cualquier triangulación de un polígono simple es 3-coloreable.

Teorema 2. [15] Sean P un polígono simple y T cualquier triangulación de P. Entonces T es 3-coloreable.

Prueba: Sea $G^*(T)$ la gráfica dual de T. Dado que cualquier diagonal de T divide a P en dos, entonces eliminar cualquier arista de $G^*(T)$ divide a esta gráfica en dos, por lo que $G^*(T)$ es un árbol, lo que implica que no existen ciclos en $G^*(T)$. Notemos que un vértice $v \in T$ correspondiente a una hoja de $G^*(T)$ tiene exactamente grado dos, v siempre existe debido a que cualquier árbol tiene al menos dos hojas. Esto implica que podemos encontrar una 3-coloración haciendo un recorrido de $G^*(T)$, tal como **DFS** [14]. El recorrido puede iniciar en cualquier vértice de $G^*(T)$ coloreando los tres vértices del triángulo correspondiente en T. Durante el recorrido se mantiene la siguiente invariante: *todos los triángulos de T correspondientes a los vértices visitados en $G^*(T)$ han sido coloreados, sin pérdida de generalidad, se usan los colores: rojo, azul y negro, además dos vértices conexos no tienen el mismo color.* Ver figura 4.3. La invariante implica que hemos encontrado una 3-coloración válida al termino del recorrido. □

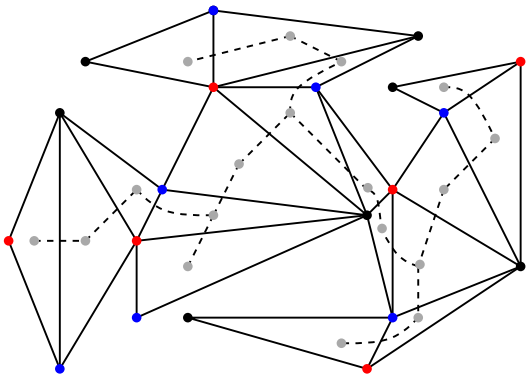
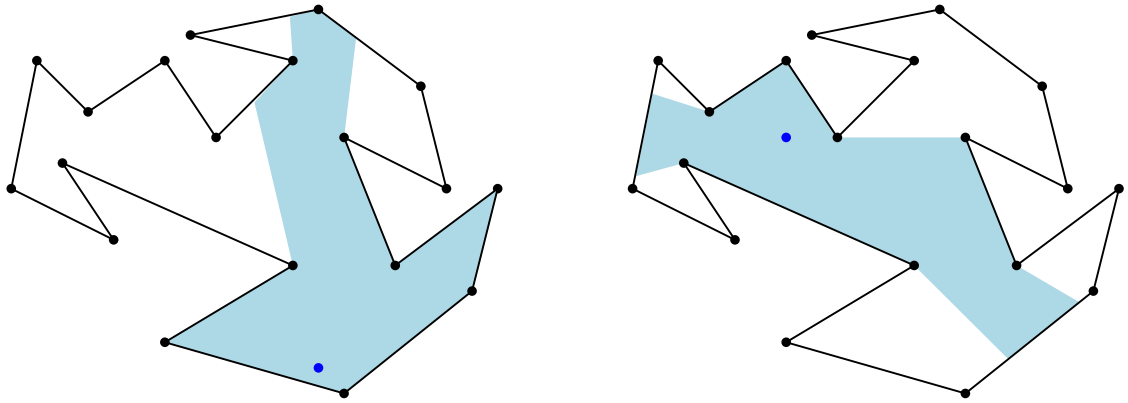


Figura 4.3: Ejemplo 3-coloración y gráfica dual

4.3. El teorema de la galería de arte

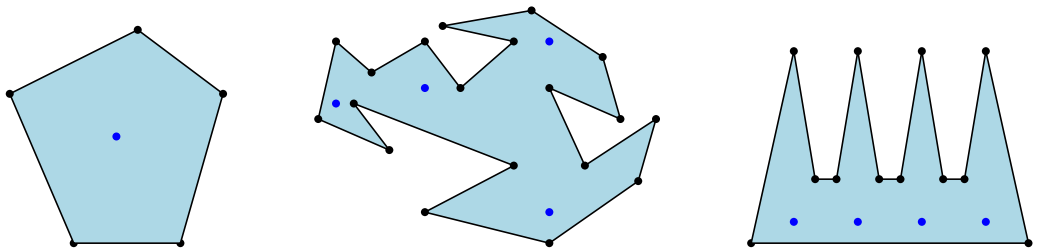
Imagina una galería de arte cuyo plano de piso es modelado como un polígono, un guardia de la galería corresponde a un punto dentro de dicho polígono. Los guardias pueden ver en cualquier dirección con un rango completo de visibilidad. Por supuesto que un guardia no puede ver a través de las paredes. Ver figura 4.4.



(a) (b)

Figura 4.4: Ejemplos del rango de visibilidad de un guardia dentro de un polígono

Definición 30. Un conjunto de guardias **cubren** o **vigilan** un polígono si cada punto en el interior del polígono es visible a algún guardia. Ver figura 4.5.



(a) (b) (c)

Figura 4.5: Ejemplo de polígonos cubiertos por un conjunto de guardias

En 1975, Chvátal [12] mostró que:

Teorema 3. ***(Chvátal)*** $\lfloor \frac{n}{3} \rfloor$ guardias son suficientes y ocasionalmente necesarios para vigilar el interior de cualquier galería de arte representada por un polígono simple de n vértices.

Unos años después, en 1978, *Fisk* [21] obtuvo una prueba más simple para el teorema de *Chvátal*, misma que se presenta a continuación.

Prueba: Sea P el polígono que representa la galería de arte y sea T una triangulación de P. Asignamos una 3-coloración a los vértices de T, esto divide en C_1, C_2 y C_3, 3 clases cromáticas, a los vértices de P. Supongamos que la cardinalidad de estas clases es a lo más $\lfloor \frac{n}{3} \rfloor$, consideremos la clase cromática de menor cardinalidad, sin pérdida de generalidad tomemos a C_1. Colocamos en cada elemento de C_1 un guardia. Dado que los vértices de cada triángulo tienen colores diferentes (rojo, azul o negro), entonces uno de ellos es de la clase cromática seleccionada, esto asegura que P queda completamente vigilado.
Ahora se mostrará que $\lfloor \frac{n}{3} \rfloor$ guardias son ocasionalmente necesarios, para esto consideremos el polígono *peine* de $n = 3m$ vértices, véase figura 4.6. Cada *pico* necesita un guardia para ser vigilado, por lo que requerimos al menos de m guardias.

□

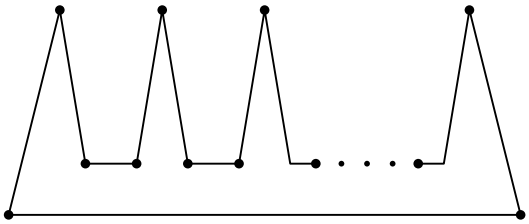


Figura 4.6: Polígono peine con $3m$ vértices

4.3.1. Algoritmo

En la sección anterior se muestra la cantidad de guardias suficientes y ocasionalmente necesarios para cubrir un polígono simple, sin embargo no se ha dicho explícitamente la forma de elegir el conjunto de guardias que cubran al polígono. En esta sección se muestra el algoritmo que obtiene un conjunto de $\lfloor \frac{n}{3} \rfloor$ guardias que cubran a un polígono simple.

Algoritmo 13: *vigilaPoligonoSimple(P)*

Data: P, un polígono simple
Result: Conjunto de guardias que vigilan todo el interior de P

```
1 begin
2     T = Triangular(P)
3     C = 3-coloracion(T)
4     Poner un guardia en cada vértice de la clase cromática de menor cardinalidad.
```

Notemos que el primer paso del algoritmo consiste en calcular una triangulación de P, esta tarea se puede realizar usando cualquier algoritmo mencionado en la sección 4.2.

El siguiente paso consiste en calcular una 3-coloración de la triangulación antes calculada. A continuación describiremos una forma de hacerlo. Sea T una triangulación de un polígono simple, comenzamos por calcular $G^*(T)$, la gráfica dual de T, es fácil ver que este paso nos tomó tiempo $O(n)$, con n la cantidad de vértices de T, pues se tienen que recorrer una sola vez todas las diagonales de T, las cuales están acotadas por $n-2$, que es la cantidad de triángulos que contiene T.
El siguiente paso es recorrer en profundidad $G^*(T)$, de tal forma que cada triángulo T representado por un vértice de $G^*(T)$ sea coloreado con tres colores distintos (s.p.g. rojo, azul y negro). El recorrido en profundidad se puede hacer usando **DFS** [14], consideremos a v un vértice de $G^*(T)$, coloreamos los vértices de t, el triángulo que representa v, luego se visitan los vecinos de v, notemos que cada triángulo representado por los vecinos de v comparten una diagonal con t, lo que implica que dos vértices de estos triángulos ya han sido coloreados, basta con colorear el vértice faltante con el color diferente a los vértices ya coloreados. Al terminar el recorrido tendremos una 3-coloración de T, lo cual toma tiempo $O(n)$ pues se tienen que visitar todos los vértices de $G^*(T)$.
Por último poner un guardia en cada vértice de un mismo color también toma tiempo $O(n)$. Entonces podemos enunciar el siguiente teorema:

Teorema 4. [44] Sea P un polígono simple con n vértices. P puede ser vigilado con a lo más $\lfloor \frac{n}{3} \rfloor$ guardias y dicho conjunto de guardias puede encontrarse en tiempo de $O(n)$.

El teorema 3 da el número de guardias suficientes y ocasionalmente necesarios para vigilar cualquier polígono simple con n vértices. Muchos polígonos pueden vigilarse con menos de $\lfloor \frac{n}{3} \rfloor$ guardias, *D. T. Lee* y *A. K. Lin* [27] probaron en 1976 que encontrar el mínimo número de guardias necesarios para vigilar un polígono es un problema *NP-completo*.

4.4. Vigilando polígonos ortogonales

Los polígonos ortogonales resultan de particular interés en los problemas de vigilancia e iluminación, debido a que en la cotidianidad la mayoría de las edificaciones que se construyen tienen una forma ortogonal, además este tipo de polígonos facilita la optimización de varios resultados del problema de vigilar la galería de arte. En específico la cota del teorema de *Chvátal* (ver sección 4.3) puede mejorarse aprovechando la estructura particular de este tipo de polígonos.

El primer resultado importante fue obtenido en 1983 por *J. Kahn, M. Klawa* y *D. Kleitman* [25], en el probaron que:

Teorema 5. Cualquier polígono ortogonal simple con n vértices puede vigilarse con a lo más $\lfloor \frac{n}{4} \rfloor$ guardias.

Antes de dar la prueba del teorema se harán algunas definiciones y se mostrarán algunos resultados.

Definición 31. Una **cuadrilaterización convexa** de un polígono ortogonal P es una partición de P en un conjunto de cuadriláteros convexos con interiores ajenos, de tal forma que las aristas del cuadrilátero son aristas de P o diagonales de P. Ver figura 4.7.

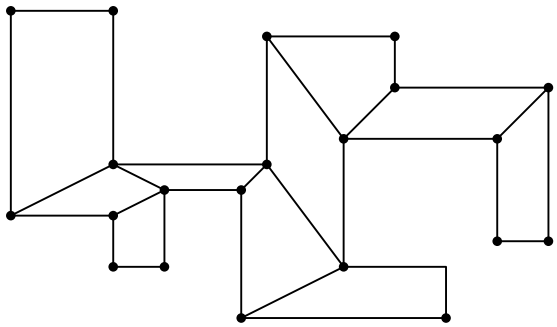


Figura 4.7: Ejemplo cuadrilaterización

Primero probaremos que cualquier polígono ortogonal puede cuadrilaterizarse convexamente, esta prueba fue dada por *A. Lubiw* [28].

Definición 32. Un polígono **1-ortogonal** P es un polígono que satisface las siguientes propiedades:

1. Todas las aristas de P, con la posible excepción de una arista distinguida e, llamada la arista sesgada, son paralelas el eje x o al eje y.
2. Con la posible excepción de e, las aristas alternan entre horizontal y vertical.
3. Todos las ángulos internos son menores o iguales que $\frac{3\pi}{2}$.
4. La nariz de la arista e no contiene vértices de P.

La nariz de e en P es el triángulo interior T con un lado horizontal, otro vertical y e como la hipotenusa, ver figura 4.8. La nariz es cerrada del lado de la hipotenusa, abierta en los otros lados: no incluye las tres esquinas. Si e es una arista horizontal o vertical entonces la nariz es vacía.

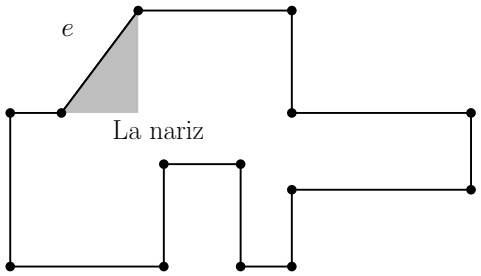


Figura 4.8: Ejemplo polígono 1-ortogonal, e y su nariz.

Teorema 6. Cualquier polígono 1-ortogonal P es convexamente cuadrilaterizable.

Prueba: Si P tiene cuatro aristas, entonces P tiene que ser convexo y el teorema es cierto para este caso. Ahora probaremos que si P tiene más de cuatro aristas entonces P siempre tiene un cuadrilátero Q' que al eliminarlo divide a P en subpolígonos 1-ortogonales más pequeños. Usando inducción se muestra el resultado. Las dos aristas incidentes a e deben ser ambas horizontales o verticales. Sin pérdida de generalidad podemos suponer que el sesgo de e está entre 0 y $\frac{\pi}{2}$ por lo que sus aristas incidentes son horizontales. Sean u y v los vértices incidentes a e tal que u es el vértice con menor coordenada y y sea w el tercer vértice de T. Lo siguiente es encontrar a Q'. Los primeros dos vértices de Q' son u y v. Sea e' el segmento de línea semi-abierto uniendo a v y w, abierto en w. Deslizamos e' hacia la derecha hasta intersectar un vértice de P. Si e' contiene una arista completa de P, entonces los vértices incidentes a esta arista, junto con u y v son los vértices de Q', ver figura 4.9. Si e' intersecta un único vértice x de P, entonces x es el tercer vértice de Q'.

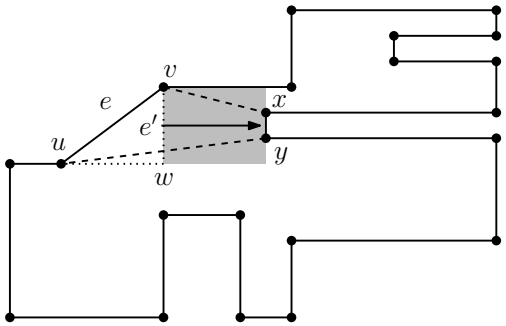


Figura 4.9: Ejemplo arista e'.

Consideremos ahora el segmento de línea semi-abierto f horizontal uniendo u a la línea vertical e' deslizada hacia la derecha como ya se ha mencionado, f no contiene a u, pero sí al vértice de la derecha. Deslizamos f hacia abajo hasta intersectar un vértice (o más) de P o una arista g de P conteniendo todo f. En el primer caso, sea y el punto más a la derecha que alcanza f. El cuarto vértice de Q' es y, ver figura 4.10

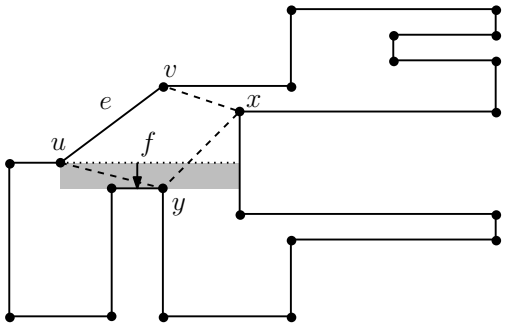


Figura 4.10: f toca al menos un vértice de P.

El caso faltante es cuando f queda contenida en la arista g de P. Esto ocurre sólo si x es

incidente a v. Consideremos el segmento de línea vertical semi-abierto h' uniendo x y g, cerrada en la parte inferior y abierta en la parte superior. Deslizamos h' a la derecha hasta alcanzar un vértice de P. Si alcanzamos más de un vértice, escogemos a y como el menor de estos puntos para ser el cuarto vértice de Q', ver figura 4.11. Así que Q' es convexo y $Q' - P$ divide a polígono en a lo más tres polígonos 1-ortogonales.

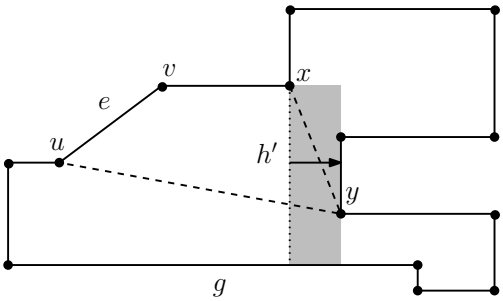


Figura 4.11: f toca una arista g de P.

□

Teorema 7. Cualquier polígono ortogonal es convexamente cuadrilaterizable.

Prueba: Notemos que un polígono ortogonal es de hecho un polígono 1-ortogonal, la definición no prohíbe que e sea paralela a uno de los ejes x o y. Entonces si elegimos cualquier arista de P como e y el teorema es consecuencia del teorema 6.

□

A continuación retomamos la demostracion del teorema 5.

Demostración. Sea P un polígono ortogonal y Q una cuadrilaterización convexa de P. Sea H la gráfica construida como sigue: el conjunto de vértices de H está formado por los vértices de Q (que son los mismos puntos que los vértices de P). El conjunto de aristas de H está formado por las aristas de Q y además dos diagonales en cada cuadrilátero $q \in Q$ conectando sus vértices opuestos, ver figura 4.12.

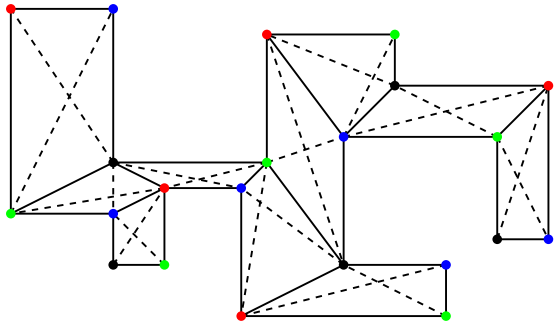


Figura 4.12: Ejemplo de polígono ortogonal, su cuadrilaterización y su gráfica H 4-coloreada.

Asignamos una 4-coloración a los vértices de H, esto divide al conjunto de vértices de P en cuatro clases cromáticas, C_1, C_2, C_3 y C_4. La cardinalidad de una de éstas es a lo más $\lfloor \frac{n}{4} \rfloor$. Notemos que cada cuadrilátero utiliza los 4 colores, esto asegura que P queda totalmente vigilado al poner en cada elemento de una de las clases cromáticas un guardia. Falta probar que H es 4-coloreable, para esto consideremos Q^* la gráfica dual de Q. Notemos que al eliminar cualquier diagonal de Q, dividimos la gráfica dual Q^* en dos subgráficas. por lo que Q^* es un árbol. Al eliminar un cuadrilátero de Q correspondiente a una hoja de Q^* se obtiene una subgráfica H' de H, la cual suponemos 4-coloreable, por lo que inductivamente puede extenderse a una 4-coloración de H. Para mostrar que $\lfloor \frac{n}{4} \rfloor$ guardias son ocasionalmente necesarios, consideremos el polígono peine ortogonal P_m con $n = 4m$ vértices mostrado en la figura 4.13. Es fácil ver que necesitamos m guardias para vigilar este polígono.

□

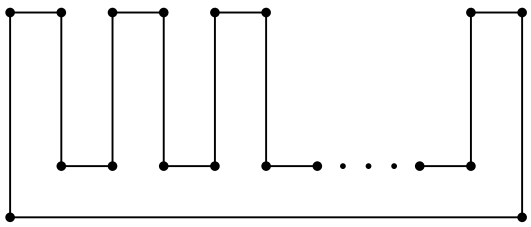


Figura 4.13: Ejemplo de polígono peine ortogonal.

4.4.1. Algoritmo

Podemos utilizar como base el algoritmo de la sección anterior para dar un algoritmo que vigila un polígono ortogonal P con n vértices.

Algoritmo 14: *vigilaPoligonoOrtogonal(P)*

Data: P, un polígono ortogonal
Result: Conjunto de guardias que vigilan todo el interior de P

```
1 begin
2     Q = Cuadrilaterizar(P)
3     H = GraficaH(Q)
4     C = 4-coloracion(H)
5     Poner un guardia en cada vértice de la clase cromática de menor cardinalidad.
```

El paso 2 obtiene a Q, una cuadrilaterización de P, para realizar este paso podemos utilizar el algoritmo de *A. Lubiw* [28], que toma tiempo $O(n \log n)$. El paso 3 obtiene la gráfica H de Q, lo cual toma tiempo $O(n)$ pues se recorren todos los cuadrilateros de Q para agregarles sus diagonales.

Para calcular la 4-coloración de H, análogamente a la 3-coloración usamos **DFS** para recorrer Q^*, la gráfica dual de Q, luego coloreamos con 4 colores distintos cada vértice de los cuadriláteros en H, esto toma tiempo $O(n)$. Por último colocar un guardia en cada vértice de la clase cromática elegida toma tiempo de $O(n)$.

Así que el algoritmo *vigilaPoligonoOrtogonal* toma tiempo de $O(n \log n)$.

4.5. Polígonos con hoyos

Si permitimos que la galería de arte contenga obstáculos en su interior, tales como pilares, paredes, etc., entonces el polígono que la representa contendrá hoyos, es decir, otros polígonos en la parte interior del polígono principal, así que $\lfloor \frac{n}{3} \rfloor$ guardias ya no son suficientes para vigilar todo el interior del polígono. En 1984, *Shermer* [29] mostró que cualquier polígono con un hoyo puede vigilarse con $\lfloor \frac{n+1}{3} \rfloor$ guardias, ésta es una cota justa, ver figura 4.14. En 1991 *Hoffman, Kaufmann* y *Kriegel* en [24] y *Sachs* y *Souvaine* en [6] de forma diferente e independiente probaron que $\lfloor \frac{n+h}{3} \rfloor$ son suficientes para vigilar cualquier polígono con h hoyos.

La idea de la prueba de *Sachs* y *Souvaine* consiste en construir un canal C que une un hoyo con la frontera del polígono P, de tal forma que solo un vértice adicional sea creado. Luego eliminamos C, lo que genera un nuevo polígono P^k con $n + k$ vértices y $h - k$ hoyos. Después de eliminar los h hoyos se puede vigialr P^k con el algoritmo *vigilaPoligonoSimple* de la sección 4.3. La única parte de P no vigilada es el canal C. Asociamos un triángulo t con C durante su construcción, con la propiedad de que cualquier punto de t puede vigilar todo C en el polígono original P. Sin embargo, los guardias que escogemos de la 3-coloración, son vértices de P^k, estos vértices no necesariamente son vértices en P y el guardia dentro del t es un guardia en un punto P.

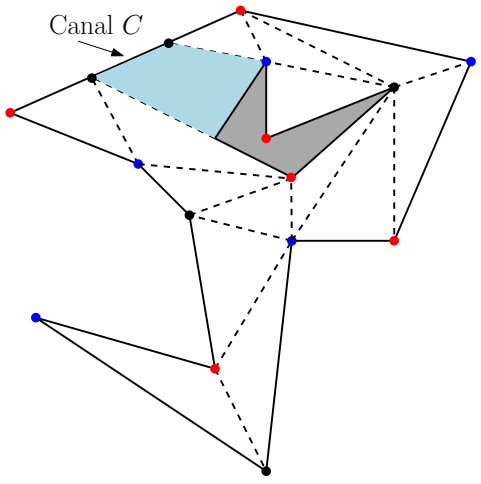


Figura 4.14: Polígono con un hoyo, su triangulación y 3-coloración

El problema de encontrar la cota justa en el número de guardias en los vértices para vigilar un polígono con n vértices y h hoyos permanece abierto, el mejor resultado conocido es de $\lfloor \frac{n+2h}{3} \rfloor$ guardias, obtenido por *O'Rourke* en [29].

4.6. Tabla de complejidades

La siguiente tabla resume varios resultados obtenidos sobre el problema de iluminación de la galería de arte:

Polígono	Algoritmo	Complejidad	Autor
Simple	Triangular	$O(n \log n)$	*M. R. Garey, D. S. Johnson, F. P. Preparata y R. E. Tarjan* 1978
Simple	Triangular	$O(n \log \log n)$	*R. E. Tarjan y C. J. Van Wyk* 1988
Simple	Triangular	$O(n)$	*B. Chazelle* 1990
Simple	Vigilar	$O(n \log n)$	*S. Fisk* 1978
Simple	Vigilar	$O(n)$	*B. Chazelle*[1]1990
Ortogonal	Cuadrilaterización convexa	$O(n \log n)$	*A. Lubiw* 1985
Ortogonal	Vigilar	$O(n \log n)$	*J. Kahn, M. Klawe y D. Kleitman* 1985

Cuadro 4.1: Tabla de complejidades.

[1]El algoritmo de *S. Fisk* utiliza la triangulación del polígono para vigilar, por lo que el algoritmo de *B. Chazelle* para triangular el polígono mejora la complejidad del algoritmo de *S. Fisk*

Capítulo 5

Iluminación con reflectores y con módems

El problema clásico de la galería de arte tal como se ha presentado en el capítulo anterior se ha generalizado en diferentes variantes. Algunas de éstas generalizaciones tienen soluciones elegantes, algunas tienen soluciones complicadas y existen varios problemas sin resolver. En este trabajo se estudian algunos resultados de las variantes: iluminación con reflectores e iluminación con módems.

5.1. Reflectores

En los problemas de iluminación o vigilancia del interior de un polígono normalmente suponemos que las fuentes de luz emiten luz en todas direcciones, es decir, iluminan con un ángulo de 360 grados. En esta sección se estudia una variante del problema iluminación en el cual nuestras fuentes de luz tienen un ángulo de iluminación restringido, tal como funciona un reflector en la realidad. El concepto de *reflector* en problemas de iluminación se introdujo por primera vez en 1994 por *P. Bose*, *L. Guibas*, *A. Lubiw*, *M. Overmars*, *D. Souvaine* y *J. Urrutia* [8].

Definición 33. Un **reflector** es una fuente de luz colocada sobre un punto p en el plano con un ángulo de iluminación α. Ver figura 5.1.

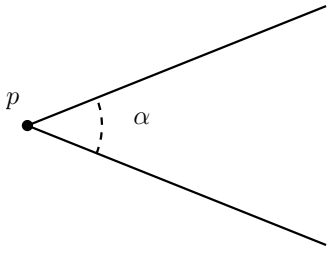


Figura 5.1: Un reflector en el plano

El primer problema estudiado sobre iluminación de reflectores, es el conocido como el problema de la *iluminación de escenarios* [13]: Supongamos un teatro con su escenario, representado por un segmento de línea s y un conjunto de reflectores $F = \{f_1, \cdots, f_n\}$ todos ubicados sobre el mismo lado de s, ¿es posible rotar los reflectores sobre su posición de tal forma que s quede completamente iluminado? Ver figura 5.2.

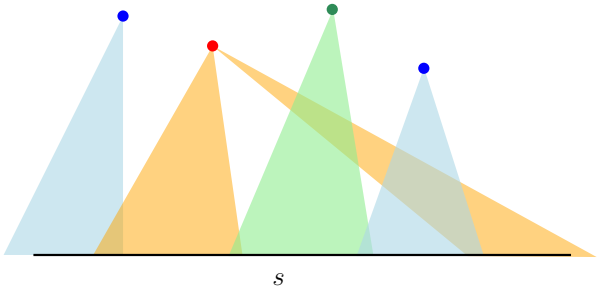


Figura 5.2: Un conjunto de reflectores que iluminan s

Dado el conjunto de reflectores F cada uno con ángulo de iluminación α_i con $i = 1, \cdots, n$, se asocia el costo angular a F:

$$\alpha(F) = \sum_{i=0}^{n} \alpha_i$$

Hasta el momento no se conoce un algoritmo eficiente que resuelva el problema. En 1998 *F. Contreras*, *J. Czyzowicz*, *E. Rivera-Campo* y *J. Urrutia* mostraron que si permitimos colocar más de un reflector en cualquier punto dado, se puede iluminar s minimizando el costo angular de F en tiempo $O(n \log n)$ [13].

5.1.1. Notación

Para facilitar la presentación de los problemas de iluminación con reflectores, en adelante si un reflector r ilumina una zona angular de tamaño α llamaremos a r un *α-reflector*, también α será llamada el *tamaño* de r. En el caso que $\alpha = \frac{\pi}{2}$, r se llamará un *reflector ortogonal*. Ver figura 5.3.
Si un reflector es colocado sobre un vértice v de un polígono, se le llamará un *v-reflector*. Ver figura 5.3.

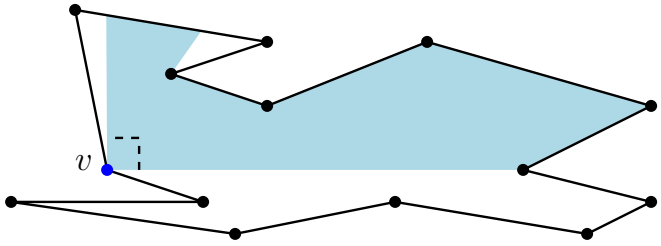


Figura 5.3: El reflector ubicado en v es un reflector ortogonal, también le podemos llamar v-reflector

5.1.2. Iluminación de polígonos convexos

En siguiente problema es mencionado en [32] el año de 1995, el cual fue propuesto y resuelto por *J. Urrutia*; publicado posteriormente en [36] en 1997. Supongamos que tenemos tres reflectores con ángulos de iluminación $\alpha_1, \alpha_2, \alpha_3$ tal que la suma de éstos es igual a π. ¿Podemos colocar estos tres reflectores con la restricción de poner solo uno por vértice de tal manera que iluminen todo el interior de cualquier polígono convexo?

Teorema 8. [36] Dado un polígono convexo P con n vértices y tres reflectores f_1, f_2, f_3 con ángulo de iluminación $\alpha_1, \alpha_2, \alpha_3$ respectivamente, tal que $\alpha_1 + \alpha_2 + \alpha_3 = \pi$, siempre es posible iluminar P posicionando los reflectores en exactamente tres vértices diferentes.

Prueba: Si $n = 3$ el teorema es cierto por las propiedades de los triángulos. Si $n > 3$, sin pérdida de generalidad supongamos que $\alpha_1 \leq \alpha_2 \leq \alpha_3$. Notemos que $\alpha_2 < \frac{\pi}{2}$ y que existe un vértice v de P cuyo ángulo interno es mayor igual a $\frac{\pi}{2}$, debido a que $n > 3$.
Ahora encontremos el triángulo T con ángulos internos $\alpha_1, \alpha_2, \alpha_3$ tal que el vértice de T con ángulo α_2 esté colocado en el vértice v y los otros vértices de T colocados en dos puntos x, y en la frontera de P. Al colocar el reflector f_2 con ángulo α_2 en v iluminando T, pueden ocurrir dos casos:

1. Los puntos x y y son de la misma arista de P, llamemos e a dicha arista. Sean u y w los vértices adyacentes de e, es fácil ver que los ángulos formados por vux y ywv son de tamaño menor o igual α_1 y α_3 respectivamente e iluminan el interior de P. Ver figura 5.4.

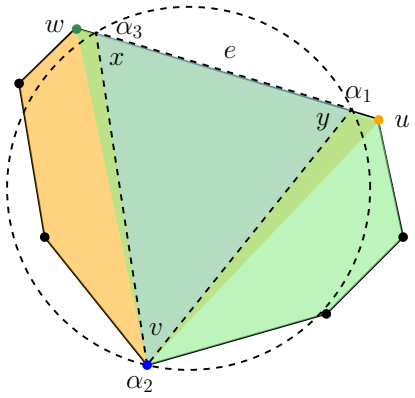


Figura 5.4: Iluminación con tres reflectores, $x, y \in e$

2. Los puntos x y y pertenecen a diferentes aristas de P. Sean e_x y e_y estas aristas. Consideremos el círculo C que circunscribe a T. Al menos un vértice adyacente a e_x y e_y no está contenido en el interior de C. Sean u y w estos vértices respectivamente. Pueden suceder dos casos:

a) $u \neq w$. Colocamos los reflectores f_1 y f_3 en los vértices u y w de tal forma que iluminen la región determinada por vux y vwy respectivamente. Así f_1, f_2 y f_3 iluminan P. Ver figura 5.5.

b) $u = w$. Consideremos las tangentes a C en x y y, es fácil verificar que el ángulo α de P en u es de tamaño a lo más $\pi - 2\alpha_2$, el cual es menor o igual que $\alpha_3 = \pi - (\alpha_1 + \alpha_2)$. Colocamos un reflector de tamaño α en u. De esta forma el polígono P queda iluminado.

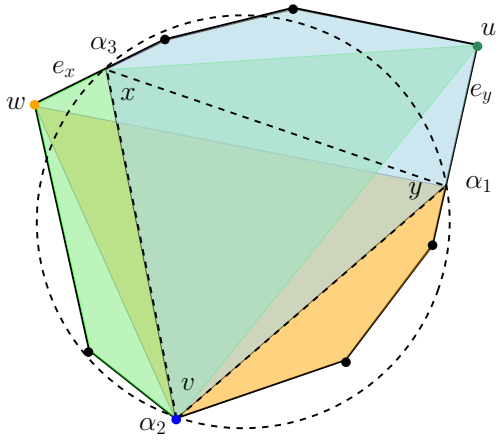


Figura 5.5: Iluminación con tres reflectores, $x \in e_x$, $y \in e_y$

□

Después de ver el resultado anterior nos preguntamos si la generalización a este problema es cierta. Dado un polígono convexo P con n vértices y un conjunto de $k \leq n$ reflectores tal que sus tamaños sumen π, ¿es posible colocar los reflectores en diferentes vértices y orientarlos de tal forma que iluminen todo P? También en 1995, *J. O'Rourke, T. Shermer* e *I. Streinu*, probaron que esta generalización es falsa [32]. En el mismo año *V. Estivil-Castro* y *J. Urrutia* probaron que siempre se puede iluminar, usando solo dos reflectores, un polígono convexo en posición circular general (i.e. con la restricción de que cuatro vértices del polígono no caigan en la frontera de un mismo círculo), de tal forma que la suma de sus tamaños sea óptima, además la respuesta a este problema se puede calcular en tiempo de $O(n^2)$ [17]. El problema de decidir si la complejidad del problema de los dos reflectores es $O(n^2)$, $O(n \log n)$ o menor permanece abierto.

5.1.3. Iluminación de polígonos simples

Consideremos a P, un polígono simple con n vértices. Notemos que si se nos permite colocar a lo más $n-2$ reflectores de tamaño menor o igual a $\frac{\pi}{3}$ sobre los vértices de P , éstos pueden ser colocados de tal manera que todo P queda iluminado. La prueba fue dada por *J. Urrutia* en el año de 1995, la cual consiste en observar que cualquier triangulación

de P contiene exactamente $n-2$ triángulos y que cada triángulo tiene un ángulo de tamaño a lo más $\frac{\pi}{3}$.
Esta solución permite que en algún vértice coloquemos más de un reflector. Entonces dado un subconjunto S de vértices de P y $\alpha(S) = \sum_{i=0}^{m} \alpha_i$ el costo angular de S:

Teorema 9. [43] Todo polígono P con n vértices tiene un subconjunto S que lo ilumina de tal manera que $\alpha(S)$ es a lo más $\frac{(n-2)\pi}{3}$. Para cada $\varepsilon > 0$ existen polígonos tales que no tienen un subconjunto de vértices que los iluminen de costo angular $\frac{(n-2)\pi}{3} - \varepsilon$.

Prueba: Recordemos que la suma de los ángulos internos de cualquier polígono con n vértices es $(n-2)\pi$. Luego triangulamos P y 3-coloreamos los vértices de P de tal forma que si dos vértices están unidos por una arista de P o una diagonal de la triangulación, reciben colores diferentes. Esto genera una partición de los vértices de P en tres subconjuntos ajenos, cada uno de los cuales iluminan P. Por lo que alguno de ellos tiene peso a lo más $\frac{(n-2)\pi}{3}$.
Para probar que $\frac{(n-2)\pi}{3} - \varepsilon$ no es suficiente, consideremos el polígono como se muestra en la figura 5.6.

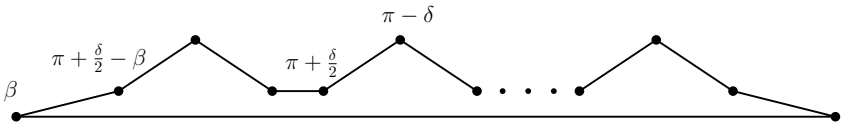


Figura 5.6: Polígono con $n = 3m + 2$ vértices

El polígono contiene $n = 3m + 2$ vértices, m vértices de tamaño $\pi - \delta$. Notemos que el polígono puede ser escogido de tal manera que cada vértice de tamaño $\pi - \delta$ es visible solo desde los vértices adyacentes a él, δ puede escogerse tan pequeño como se quiera. Luego para iluminar dichos puntos necesitamos elegir un vértice de tamaño al menos $\pi - \delta$, entonces para iluminar P necesitamos un subconjunto de vértices de peso al menos $m(\pi - \delta)$.

□

Otra pregunta que surge sobre la iluminación de polígonos con reflectores es la siguiente: ¿existe un $\alpha < \pi$ tal que todo polígono quede iluminado colocando en cada vértice un reflector de tamaño a lo más α?
La respuesta a esta pregunta fue dada el año de 1995 por *V. Estivill-Castro*, *J. O'Rourke*, *J. Urrutia* y *D. Xu* en [18]. Sea P un polígono simple con n vértices, es fácil ver que colocando un reflector de tamaño a lo más π en cada vértice, P puede ser iluminado. Para probar esto, consideremos una triangulación de P con $n-3$ diagonales, observemos que una de estas diagonales que une dos vértices u y v corta a P en dos partes, una de las cuales es un triángulo con vértices u, v, z. Coloquemos en z un reflector de tamaño igual a z de tal manera que quede iluminado el triángulo uvz. Este resultado puede probarse inductivamente sobre el número de vértices de P.

La mejor cota inferior para iluminar polígonos simples con π-reflectores, $\lfloor\frac{(n-1)}{2}\rfloor$, fue dada por *J. O'Rourke* el año de 1997 en [31], esta cota es cierta para la familia de polígonos llamada *montañas monotonas* (polígonos con dos cadenas de vértices reflejo, ver definición 34).

Definición 34. Un **vértice reflejo** es un vértice de un polígono tal que el ángulo formado en el interior del polígono por las aristas incidentes a él es mayor a 180 grados.

Un π-reflector puesto en un vértice reflejo no puede iluminar completamente el área angular acotada por las aristas incidentes a dicho vértice. Si se restringe la orientación del π-reflector sobre el vértice reflejo se tienen tres variantes del problema de iluminar el interior del polígono con π-reflectores. Sea P un polígono simple, si cada reflector v ubicado en un vértice reflejo de P tiene la misma orientación, las variantes del problema son (ver figura 5.7):

- *inward-facing*: ninguna de las aristas incidentes a v se intersectan con la frontera de la zona angular del π-reflector.
- *edge-aligned*: alguna de las aristas incidentes a v es colinear con alguna de las rectas que delimitan la zona angular del π-reflector.
- *general*: no hay ninguna restricción en la orientación del π-reflector.

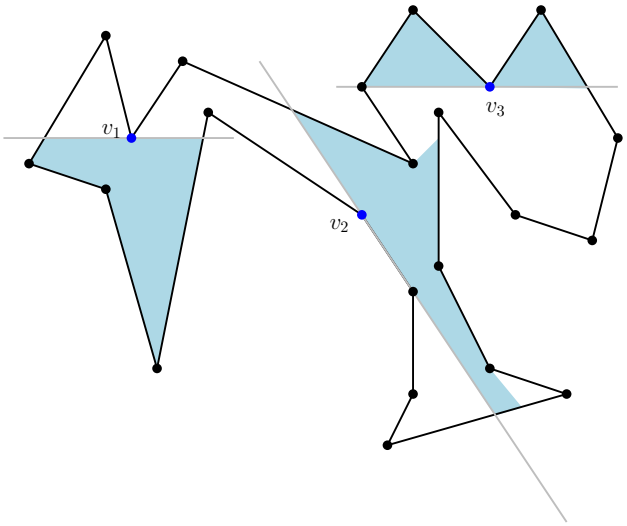


Figura 5.7: v_1 vértice *inward-facing*, v_2 vértice *edge-aligned*, v_3 vértice *general*

En las variantes de *inward-facing* o *edge-aligned* la cota dada en [18] fue la mejor cota superior hasta el año 2005 cuando *B. Speckmann* y *C. D. Tóth* en [37] obtuvieron la cota de $\lfloor\frac{(2n-k)}{3}\rfloor$ *edge-aligned* π-reflectores para iluminar cualquier polígono simple. *B. Speckmann* y *C. D. Tóth* consideran que es la primera cota no trivial y justa para el peor caso de familias de polígonos conocidas hasta el momento de la publicación del resultado.

Para la variante de π-reflectores con orientación *general* en el año 2001 *C. D. Tóth* en [40] muestra que se puede iluminar cualquier polígono simple con a lo más $\lfloor \frac{(3n-5)}{4} \rfloor$ π-reflectores. Posteriormente en el mismo año, *V. Brumberg*, *S. Ramaswami* y *D. Souvaine* en [9] dieron de forma simple la cota de $\lfloor \frac{5n}{6} \rfloor$ π-reflectores. Tiempo después, en el 2005 *B. Speckmann* y *C. D. Tóth* mostraron que cualquier polígono simple puede ser iluminado por a lo más $\lfloor \frac{n}{2} \rfloor$ π-reflectores [37].

Hasta este punto se han mencionado resultados sobre iluminación de polígonos utilizando π-reflectores, otra pregunta que surge es la siguiente: ¿se puede iluminar el interior del polígono usando reflectores de tamaño menor a π? En 2003 *C. D. Tóth* mostró que el mínimo número de $\frac{\pi}{4}$-reflectores para iluminar el interior de un polígono con n vértices es $n-1$ si n es impar y $n-2$ si n es par [41].

5.1.4. Iluminación de polígonos ortogonales

El primer resultado de iluminación de polígonos ortogonales usando reflectores fue obtenido por *V. Estivil-Castro* y *J. Urrutia* el año de 1994 en [19].

Teorema 10. Cualquier polígono ortogonal con n vértices siempre puede ser iluminado usando a lo más $\lfloor \frac{3n-4}{8} \rfloor$ $\frac{\pi}{2}$-reflectores.

Prueba: Sea P el polígono ortogonal, todos los vértices de P forman ángulos interiores de tamaño $\frac{\pi}{2}$ o $\frac{3\pi}{2}$. La suma de los ángulos interiores de un polígono simple de n vértices es igual a $(n-2)\pi$, sean c y r los vértices convexos y cóncavos de P respectivamente, es claro que $n = c + r$. Así que $(n-2)\pi = c\frac{\pi}{2} + r\frac{3\pi}{2}$, resolviendo para c y sustituyendo en $n = c + r$, tenemos que $c = \frac{n+4}{2}$, análogamente para r tenemos: $r = \frac{n-4}{2}$ [29].
Una arista e de P se llamara *superior*, si el interior de P está por debajo de e. En forma análoga, podemos definir aristas *derechas*, *izquierdas* e *inferiores* de P. Resulta fácil ver que P puede iluminarse con $\frac{\pi}{2}$-reflectores utilizando la siguiente regla de iluminación, ver figura 5.8.

Regla izquierda-superior:

a) En el extremo superior de toda arista *izquierda*, colocamos un $\frac{\pi}{2}$-reflector de tal forma que ilumine el sector angular entre $\frac{3\pi}{2}$ y 2π.

b) En el extremo izquierdo de toda arista *superior*, colocamos un $\frac{\pi}{2}$-reflector de tal forma que ilumine el sector angular entre $\frac{3\pi}{2}$ y 2π.

Análogamente podemos definir tres reglas más de iluminación, *superior-derecha*, *regla derecha-inferior* y *regla inferior-izquierda*.
Así que podemos iluminar P de cuatro formas distintas colocando $\frac{\pi}{2}$-reflectores en los vértices de P. Notemos que los conjuntos de reflectores utilizados en cada una de las reglas de iluminación son ajenos. Si ponemos $\frac{\pi}{2}$-reflectores usando las cuatro reglas simultáneamente, en cada vértice cóncavo pondremos exactamente dos $\frac{\pi}{2}$-reflectores y en cada vértice convexo uno.

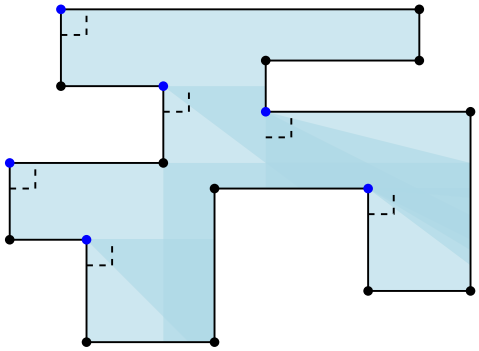


Figura 5.8: Regla izquierda-superior

Como el número de vértices cóncavos es $\frac{n-4}{2}$ y el número de vértices convexos es $\frac{n+4}{2}$, el número total de reflectores utilizado por las cuatro reglas es $\frac{3n-4}{2}$, esto indica que una de las reglas de iluminación utiliza a lo más $\lfloor \frac{3n-4}{8} \rfloor$ $\frac{\pi}{2}$-reflectores.
Para probar la existencia de polígonos ortogonales que requieren $\lfloor \frac{3n-4}{8} \rfloor$ $\frac{\pi}{2}$-reflectores, consideremos los polígonos de la figura 5.9.

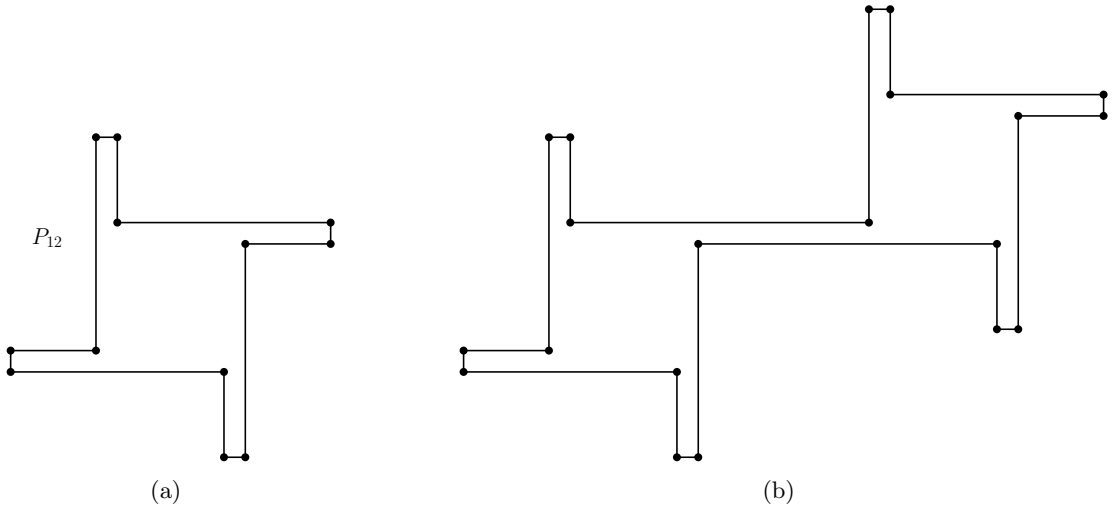


Figura 5.9: (a) Polígono hélice P_{12}, (b) Polígono hélice resultado de unir dos polígonos P_{12}

El primero etiquetado como P_{12} llamado la *hélice* con cuatro aspas tiene doce vértices y requiere cuatro $\frac{\pi}{2}$-reflectores. El segundo se obtiene pegando dos hélices por una aspa. Es fácil ver que el número de vértices incrementa en ocho y el número de reflectores requeridos incrementa en tres. Si repetimos la operación de pegar copias extra del polígono hélice se puede obtener una familia de polígonos con $n = 12 + 8m$ vértices que requieren $4 + 3m$ $\frac{\pi}{2}$-reflectores, lo cual prueba la cota inferior.

□

Posteriormente en 1998, se obtuvo el siguiente resultado:

Teorema 11. Todo polígono ortogonal P con h hoyos y n vértices puede iluminarse con a lo más $\lfloor \frac{3n+4(h-1)}{8} \rfloor$ $\frac{\pi}{2}$-reflectores colocados sobre los vértices de P.

La prueba de este teorema se puede consultar ampliamente en [1] y [43].

5.2. Módems

En esta sección se estudia una nueva variante del problema de iluminación llamada *el problema de iluminación con modems*. Este problema deriva del uso diario de computadoras portátiles y módems inalámbricos. La experiencia demuestra que al intentar conectar una computadora portátil a un módem inalámbrico, hay dos factores que deben tomarse en cuenta: la distancia entre la computadora y el módem y quizás lo más importante para muchas construcciones, el número de paredes que separan nuestra computadora del módem.

Esta variante fue introducida por *O. Aichholzer*, *R. Fabila-Monroy*, *D. Flores-Peñaloza*, *T. Hackl*, *J. Urrutia* y *B. Vogtenhuber* en [2] en el 2009.

Definición 35. Un **k-módem** es un módem omnidireccional inalámbrico colocado sobre un punto p en el plano, con la capacidad de transmitir una señal estable a través de k paredes a lo largo de una línea recta. Ver figura 5.10.

Las paredes se representan teóricamente como líneas, segmentos de línea o aristas de polígonos. Podemos decir que k es el *poder* del módem. Una vez definido el k-modem, se define el concepto de visibilidad o iluminación bajo este tipo de fuentes.

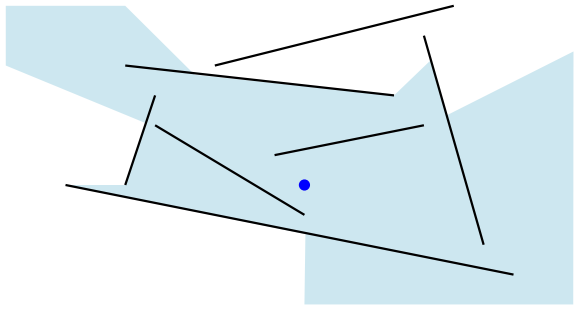


Figura 5.10: Ejemplo de un 1-módem iluminando un conjunto de segmentos

Definición 36. Para una k positiva, un punto p dentro de un polígono P es iluminado por un k-módem m si el segmento de línea que une a p con m cruza a lo más k aristas (*paredes*) de P. Ver figura 5.11

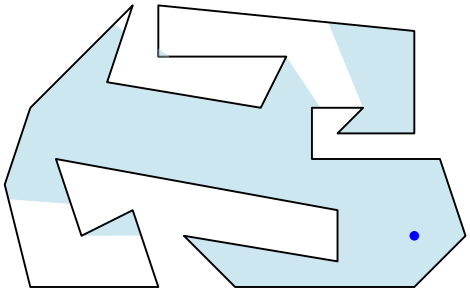


Figura 5.11: Ejemplo de un 2-módem iluminando el interior de un polígono

5.2.1. El problema de iluminación con módems

Problema 1. [20] **El problema de iluminación con módems:** ¿Cuántos k-módems son siempre suficientes para iluminar todo el interior de un polígono simple P con n vértices? Ver figura 5.12

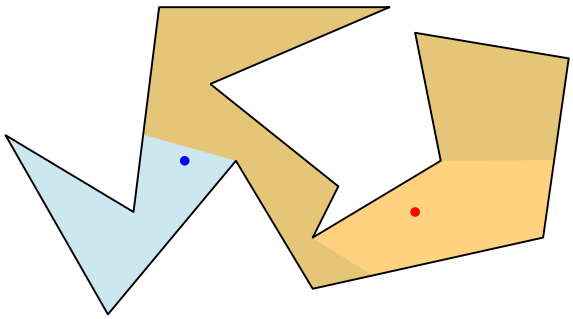


Figura 5.12: Polígono iluminado por dos 2-módems, uno no es suficiente.

El primer resultado sobre esta variante se obtuvo el año 2009 en una versión previa de [2] presentada durante la *EuroCG'09*, donde se muestra que todo polígono monótono con n vértices puede ser iluminado con $\lceil \frac{n}{2k} \rceil$ k-módems y se exhibe una familia de polígonos monótonos ortogonales que requieren $\lceil \frac{n}{2k+2} \rceil$ k-módems para ser iluminados. Ver figura 5.13

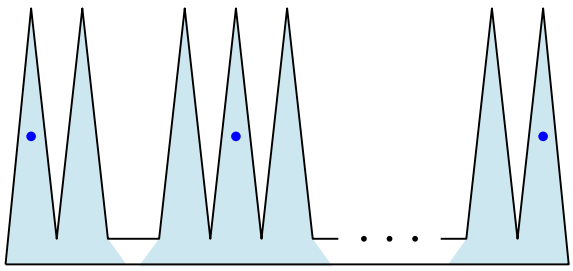


Figura 5.13: Polígonos que requieren $\frac{n}{2k+2}$ k-módems.

En 2017 esta cota fue mejorada mostrando que cualquier polígono monótono con n vértices

se puede iluminar con $\lceil \frac{n-2}{2k+3} \rceil$ k-módems y de igual forma exhibe una familia de polígonos monótonos ortogonales que requieren $\lceil \frac{n-2}{2k+3} \rceil$ k-módems.
En 2009 *R. Fabila-Monroy*, *A. Ruiz Vargas* y *J. Urrutia* probaron los siguientes resultados [20]:

Teorema 12. Sea P un polígono ortogonal con $2m$ aristas. Si m es par, P siempre puede ser vigilado con un $(m-1)$-módem ubicado en el interior de P. Si m es impar, un m-módem siempre es suficiente para iluminar P.

La prueba se sigue del siguiente resultado: Sea $p = (a, b)$ un punto en el plano, definimos $C_+^+(p) = \{q = (x, y) : a \leq x, b \leq y\}$. Si p es el origen $C_+^+(p)$ es el cuadrante positivo del plano. Análogamente se define $C_+^-(p) = \{q = (x, y) : a \leq x, b \geq y\}$, $C_-^-(p) = \{q = (x, y) : a \geq x, b \geq y\}$ y $C_-^+(p) = \{q = (x, y) : a \geq x, b \leq y\}$.

Lema 1. Sea P un polígono ortogonal y p un punto en el plano. Supongamos que k vértices de P pertenecen al interior de $C_+^+(p)$. Entonces cualquier rayo lanzado desde p contenido en $C_+^+(p)$ cruza a lo más k aristas de P.

Prueba: Sea $p^{\rightarrow}$ un rayo contenido en $C_+^+(p)$ que es lanzado desde p. Si $p^{\rightarrow}$ intersecta una arista horizontal e de P desplazamos esta intersección al vértice derecho de e. Si $p^{\rightarrow}$ intersecta una arista vertical f de P desplazamos esta intersección al vértice superior de f. Podemos imaginar esto como dar una orientación a las aristas horizontales de izquierda a derecha y de arriba hacia abajo a las aristas verticales. Resulta fácil ver que cualquier rayo $p^{\rightarrow}$ puede desplazar a lo más una intersección a cada vértice de P contenido en $C_+^+(p)$.
□

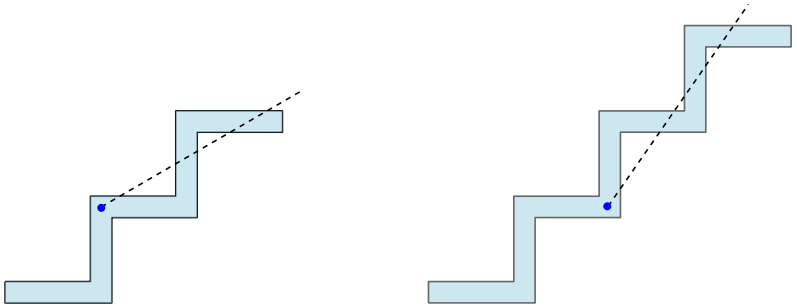
Claramente el lema es cierto para $C_+^-(p)$, $C_-^-(p)$ y $C_-^+(p)$. Ahora probaremos el teorema: 12.

Prueba: Supongamos que m es par, sea ℓ la línea que deja $\frac{m}{2}$ aristas horizontales de P en interior de cada uno de los semiplanos definidos. Entonces P tiene exactamente m vértices arriba de ℓ. Tomamos un punto p en el interior de P contenido en ℓ. Resulta fácil ver que, dado que p pertenece al interior de P, cada semiplano C_+^+, $C_+^-(p)$, $C_-^-(p)$ y $C_-^+(p)$ contiene a lo más $m - 1$ vértices en P. Por el lema anterior se prueba la cota superior. Ver figura 5.14. Para el caso cuando m es impar se muestra de forma similar.

□

También en [20] se menciona que cualquier polígono ortogonal P con n aristas puede ser iluminado con un $\lceil \frac{n}{3} \rceil$-módem, para lo cual se da un bosquejo de la prueba. Se muestra también que cualquier polígono simple P con n aristas siempre puede ser iluminado con un módem de poder $\lceil \frac{2n+1}{3} \rceil$. La prueba de este último resultado se obtiene directamente del teorema de *Fulek*, *Holsem* y *Pach* [22], el cual da una cota para la cantidad de objetos que intersecta un rayo en una familia de conjuntos convexos.

En el 2010 en [5] se muestra que $\frac{n}{6}$ 2-módems son a veces necesarios para iluminar

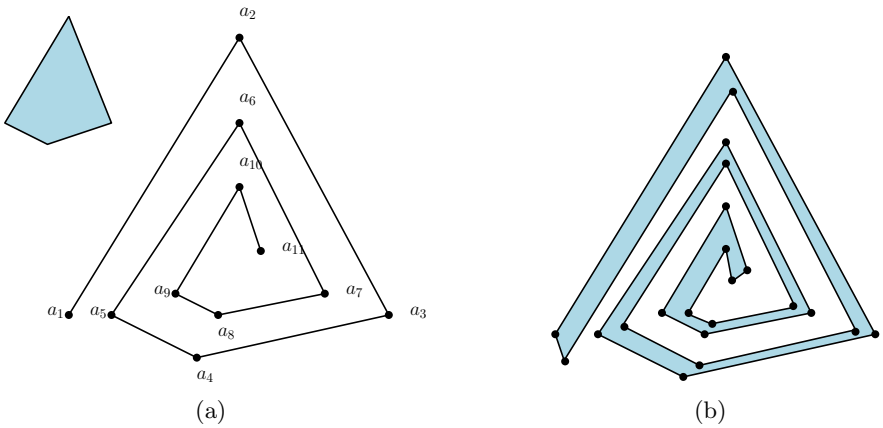


(a) (b)

Figura 5.14: Ejemplo de polígonos que requieren un $(m-1)$-módem para ser iluminado.

el interior de cualquier polígono simple. Y se introduce una clase de polígonos espiral llamada *spirangles*, ver definición 37.
Dos aristas son *homotéticas* si una arista es una imagen escalada y trasladada de la otra. Un *t-spirangle* es una cadena poligonal $A = a_1, a_2, \cdots, a_m$ que forma un espiral hacia dentro alrededor de un punto central tal que cada t aristas completa un giro de 2π y cada par de aristas (a_i, a_{i+1}), (a_{i+t}, a_{i+1+t}) son homotéticas, para $1 \leq i \leq m-t$. Se puede suponer que el giro del espiral es en sentido horario.

Definición 37. Un polígono ***spirangle*** *homotético* P es un polígono simple tal que su frontera consiste de dos cadenas *t-spirangle* anidadas $A = a_1, a_2, \cdots, a_m$ y $B = b_1, b_2, \cdots, b_m$ más dos aristas adicionales (a_1, b_1) y (a_m, b_m) que unen los puntos extremos de A y B. Ver figura 5.15.



(a) (b)

Figura 5.15: (a)La cadena A y el polígono convexo que la genera. (b) Polígono spirangle homotético.

Sea P un polígono spirangle, se muestra que $\lfloor \frac{n}{8} \rfloor$ 2-módems son suficientes y algunas veces necesarios para iluminar el interior de P. Para el caso de polígonos espirales arbitrarios

(polígonos formados por cadenas poligonales convexas que se reflejan unidas por sus puntos finales), se obtiene la cota de $\lfloor \frac{n}{4} \rfloor$ 2-módems suficientes para iluminar el interior de cualquier polígono [5].
Por último en el año 2016 *F. Duque* y *C. Hidalgo-Toscano* probaron en [16] que el número de k-módems requerido para iluminar cualquier polígono con n vértices es de $O(\frac{n}{k})$. Y para polígonos ortogonales se requieren $\frac{4n}{k} + \frac{5}{3}$ k-módems. Aún más, el algoritmo para encontrar los k-módems que cumplen dichas cotas toma tiempo $O(n \log n)$.

5.2.2. Algoritmo para obtener el polígono de visibilidad de un k-módem

Sea P un polígono simple con n vértices $v_1, v_2, \cdots, v_n$ y q un punto de P donde se ha ubicado un k-módem. En esta sección se muestra un algoritmo para calcular el polígono de visibilidad Q del k-módem atravesando a lo más k paredes. Este polígono tendrá zonas del interior de P y zonas del exterior. Por simplicidad consideramos P contenido en una caja rectangular R.
Este algoritmo fue dado por *A. L. Bajuelos*, *S. Canales*, *G. Hernández* y *A. M. Martins* en [4]. Bajo el supuesto de que q está en posición general y los vértices de P están ordenados en sentido antihorario definimos los *vértices críticos*:

Definición 38. Un vértice v de P es **crítico** para un punto q, con q ubicado en el interior de P, si los vértices anterior y posterior a v se encuentran en el mismo semiplano respecto de la recta vq. Ver figura 5.16.

El algoritmo se basa en la técnica de *barrido de línea angular*, comenzamos por trazar rectas con origen en q y que pasan por los vértices críticos de P para q. Luego desde q se ordenan angularmente dichas rectas que servirán como eventos de nuestro barrido angular.

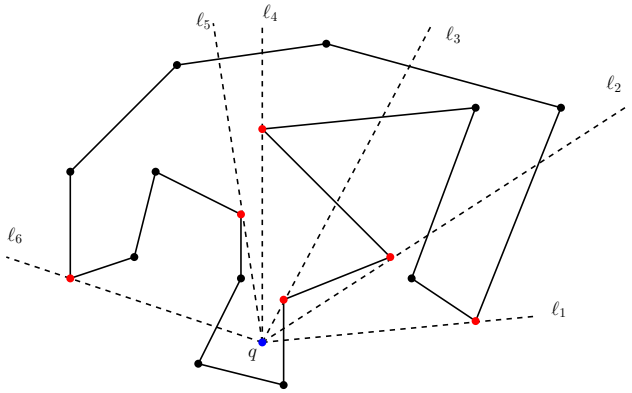


Figura 5.16: Vértices críticos y el conjunto de rectas con origen en q ordenadas angularmente.

El siguiente paso consiste en trazar el borde de Q. Para esto primero detectamos un punto z sobre dicho borde, sin pérdida de generalidad tomemos el rayo con origen en q y dirección hacia la derecha de este punto y localizamos el punto de intersección z con P después de

atravesar k paredes. Si se alcanza el rectángulo R antes de atravesar k paredes tomamos z como el punto de intersección entre el rayo y R. A partir de z se comienza a trazar el borde de Q. Si k es par avanzamos en sentido antihorario por la frontera de P hasta alcanzar la siguiente recta con origen en q. Si k es impar avanza en sentido horario por la frontera de P hasta alcanzar la siguiente recta.
En cada recta correspondiente a un evento, si t es el punto de intersección entre la recta evento y la frontera de P sobre la que se ha avanzado, tenemos dos posibles casos:

(i) Si t está más cerca de q que el vértice crítico, se sigue avanzando por la frontera de P.

(ii) Si t está más lejos de q que el vértice crítico, entonces avanzamos por la recta del evento según los casos en la figura 5.17, ya sea hacia el punto crítico (casos en (a)) o en sentido contrario (casos en (b)). El siguiente vértice del polígono de visibilidad Q es el punto crítico para los casos en (a) o el segundo punto posterior que intersecta la recta del evento con P.

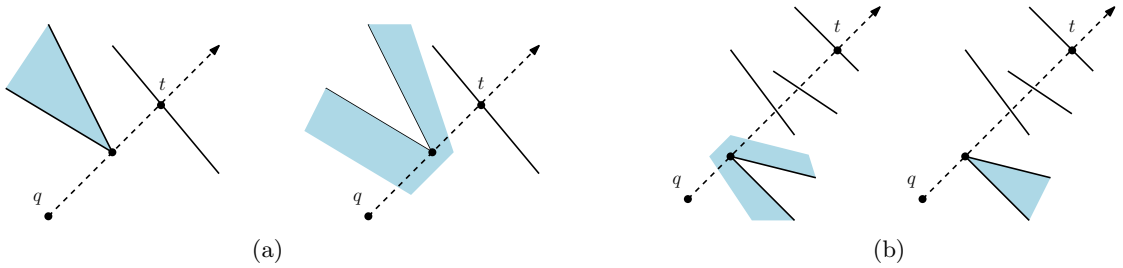


Figura 5.17: (a) Casos 1 y 2, (b) Casos 3 y 4

Por último se continúa con la construcción de Q siguiendo la nueva arista hasta alcanzar la siguiente recta. Se repite el procedimiento anterior hasta cerrar Q. Ver figura 5.18

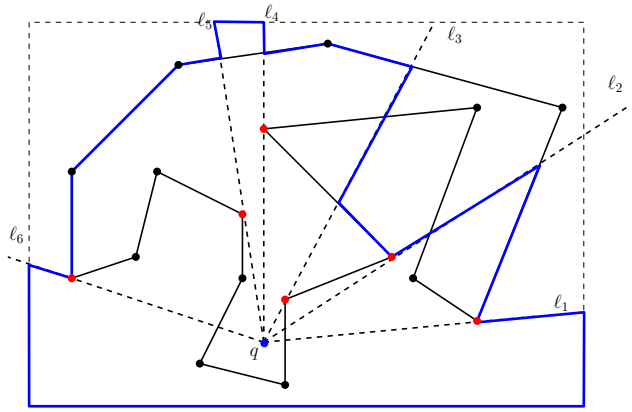


Figura 5.18: Ejemplo de la ruta para construir el polígono de visibilidad de un 2-módem

El algoritmo *PolígonoK-visibilidad* queda de la siguiente forma:

Algoritmo 15: *PoligonoK-visibilidad(P, p_m, k)*

Data: P, un polígono simple, p_m, punto donde se ubica el k-módem, k, poder del módem,
Result: Q, el polígono de k-visibilidad de p_m sobre P

```
1  begin
2      Q = ∅
3      h = ∅
4      V_c = verticesCriticos(P, p_m)            // conjunto de vértices críticos
5
6      V_c = ordenAngular(V_c)
7      (z, a_z) = calculaPuntoInicio(P, p_m, k)   // obtenemos el primer punto
       sobre la frontera de Q y la arista de P que lo contiene
8
9      inserta(z)                                 // agrega z como vértice de Q
10
11     for v_i ∈ V_c do
12         ℓ_i = recta con origen en p_m y pasa por v_i
13         t = a_z ∩ ℓ_i
14         while t = ∅ do
15             if k es par then
16                 z = siguiente(z)
17                 a_z = siguiente(a_z)
18                 inserta(z)
19             else
20                 z = anterior(z)
21                 a_z = anterior(a_z)
22                 inserta(z)
23         if distancia(t, p_m) > distancia(t, v_i) then
24             inserta(t)
25             (z, a_z) = actualizaInterseccion(ℓ_i, h)   // avanzar por ℓ_i en sentido
               contrario a v_i dos intersecciones o hasta llegar a R
26
27             inserta(z)
28         else
29             continue
30     return Q
```

Las funciones que a continuación se describen, muestran la forma en que se resolvió en este trabajo la implementación del algoritmo, puesto que al definir el algoritmo en [4], estas subrutinas se dan por entendidas y no se explican a detalle.
Algunas de estas funciones son muy intuitivas, por ejemplo: es claro que la función *distancia* calcula la distancia entre dos puntos, la función *inserta* agrega un nuevo vértice a Q y que *siguiente* y *anterior* regresan el vértice siguiente y anterior respectivamente, ésto con referencia a la orientación del polígono (horario, antihorario).
En [4] implícitamente se visitan los vértices no críticos de P, por lo que los pasos 14-22, se han agregado para construir la frontera de Q en los sectores formados por dos rectas de barrido, esto es, se sigue la frontera de P sobre sus aristas ubicadas a k intersecciones del k-módem.
La función *verticesCriticos(P, p_m)*, recorre los vértices de P y obtiene los vértices que cumplen con la definición de vértice crítico. Sea p_m el punto donde se ubica el módem y sea $v_i \in V(P)$, para determinar si un v_i es crítico, se obtiene el segmento dirigido $s_i = (p_m, v_i)$ con p_m el origen y v_i el punto final de s_i. Luego obtenemos v_{i+1} y v_{i-1}, los vértices vecinos de v_i y utilizando la función *giro*, descrita en la sección 3.2, que determina de qué lado de una recta dirigida se encuentra un punto, podemos decidir si v_{i+1} y v_{i-1} se encuentran en el mismo semiplano definido por la recta que contiene a s_i. Ver algoritmo 16.

Algoritmo 16: *verticesCriticos(P, p_m)*

```
   Data: P, un polígono simple, p_m, punto donde se ubica el k-módem
   Result: V_c, lista de vértices críticos de P
1  begin
2      for v_i ∈ V(P) do
3          s = (p_m, v_i) if giro(s, v_{i-1}) = giro(s, v_{i+1}) then
4              V_c = V_c ∪ v_i
5          else
6              continue
7      return V_c
```

Sea S un conjunto de puntos en posición general, la función *ordenAngular*, como su nombre lo dice, ordena angularmente a S. Para dar un orden angular se necesita una dirección de referencia, esta dirección puede darse por una segmento dirigido $s = (i, f)$, donde i es el punto inicial y f el punto final de s. Dicha dirección será el origen y a partir de ella se puede medir el ángulo formado por s y el segmento formado por $s_i = (i, p_i)$, con $p_i \in S$. Notemos que para obtener dichos ángulos se tendrían que usar algunas funciones trigonométricas inversas, lo que puede ocasionar pérdida de precisión en los cálculos. En este trabajo se usó una alternativa más eficiente, para ordenar angularmente los puntos no es necesario calcular los ángulos mencionados, basta con poder comparar dos puntos entre sí. La comparación que nos interesa es: si el segmento dirigido $s_i = (i, p_i)$ se encuentra a la izquierda o a la derecha del segmento $s_j = (i, p_j)$ con $i \neq j$. Para esto también se

puede utilizar la función *giro*. El algoritmo *ordenAngular*, es un *merge sort* modificado, que utiliza la comparación antes descrita para ordenar angularmente.
Consideremos p_0, igual al p_m recorrido una unidad sobre el eje X. La función *calculaPuntoInicio(P, p_m, k)*, calcula la k-ésima intersección de la semirrecta horizontal ℓ con origen en p_m y que pasa por p_0, con las aristas de P.

Algoritmo 17: *calculaPuntoInicio(P, p_m, k)*

Data: P, un polígono simple, p_m, punto donde se ubica el k-módem, k, poder del módem, h, el estado del barrido
Result: (z, a_z), donde z es el k-ésimo punto de intersección y a_z la arista que contiene a z

```
1  begin
2      ℓ = semirrecta p_m p_0
3      for a_i ∈ A(P) do
4          if z = ℓ ∩ a_i then
5              h = h ∪ (z, a_i)
6          else
7              continue
8      h = ordena(h)                    // ordena con respecto a los valores de z
9
10     return (h[k], )
```

Por último la función *actualizaInterseccion*, se mueve sobre la recta de barrido ℓ_i hacia arriba o hacia abajo, según sea el caso, ver figura 5.17. Sean v_{i+1} y v_{i-1} los vecinos del vértice v_i, dado que el algoritmo ilumina partes del interior y del exterior del polígono sin distinción, estos casos se pueden reducir a dos: el primero donde v_{i+1} y v_{i-1} están a la derecha de ℓ_i y el segundo donde v_{i+1} y v_{i-1} están a la izquierda de ℓ_i. Es fácil ver en el primer caso que las aristas $a_i = (v_i, v_{i-1})$ y $a_{i+1} = (v_i, v_{i+1})$ quedan *atrás* de la recta de barrido, es decir, estas dos aristas ya no intervienen en visibilidad del k-módem, por lo que se debe agregar a Q la intersección con la última arista a_z y calcular la nueva intersección de ℓ_i con las aristas correspondientes, moviendo t dos intersecciones más *afuera* de ℓ_i. De forma análoga, en el segundo caso, a_i y a_{i+1} están por entrar al rango de visibilidad del k-módem, por lo que se tiene que agregar a Q la intersección de ℓ_i con la última a_z y calcular la nueva intersección de ℓ_i con las aristas correspondientes, moviendo dos intersecciones hacia el origen de ℓ_i.

Complejidad

La complejidad del algoritmo se puede analizar como sigue: la línea 4 toma tiempo $O(n)$, pues basta con recorrer los vértices de P realizando una comparación para clasificar cada vértice, la tareas más costosas son el paso 6, en el cual se ordena el conjunto de vértices críticos, lo cual toma tiempo de $O(n \log n)$, y el paso 7 que también toma tiempo $O(n \log n)$, pues la función *calculaPuntoInicio* recorre todas las aristas de P y ordena la

lista de intersecciones. Luego en cada iteración del ciclo en la línea 11 se realiza un número constante de operaciones, pues cada arista de P interviene a lo más $O(1)$ veces.
Por lo que la complejidad del algoritmo *PoligonoK-visibilidad* es de $O(n \log n)$.

5.2.3. θ_k-módems

En las secciones anteriores se muestran varios resultados obtenidos en el estudio de las variantes del problema de iluminación utilizando reflectores y módems. Una pregunta que surge naturalmente es: ¿podemos combinar las características de estos dos tipos de fuentes de iluminación en una? Estas fuentes al igual que un k-módem tendrán la capacidad de atravesar k paredes, pero con un ángulo de visión de tamaño θ. A estas fuentes les llamaremos θ_k-*módem*, ver Definición: 39.

Definición 39. Un **θ_k-módem** es un módem con poder k y un ángulo de visión de tamaño θ. Ver figura 5.19.

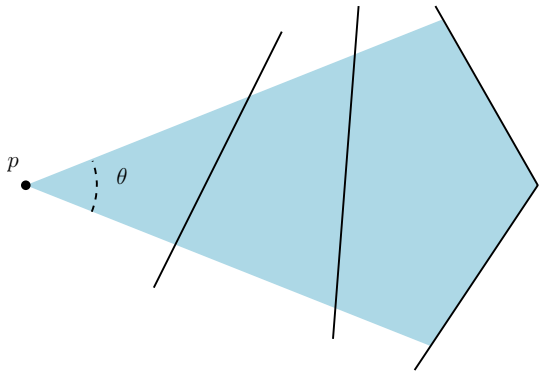


Figura 5.19: Ejemplo θ_k-módem, de poder $k = 2$.

Entonces una variante implícitamente definida por el problema de iluminación con módems es el problema de iluminación con θ_k-módems:

Problema 2. ¿Cuántos θ_k-módems son suficientes para iluminar el interior de un polígono con n vértices?

Hasta el momento no se conoce ningún resultado para este problema, por lo que se puede iniciar una nueva línea de investigación a partir del mismo.

Capítulo 6

LitPolygons, una aplicación para iluminar polígonos

6.1. Introducción

Como se ha mostrado en los capítulos anteriores, el estudio del problema de vigilar o iluminar la galería de arte es de particular interés en las Ciencias de la computación. Desde el planteamiento del problema hasta la fecha han surgido diversas variantes del mismo, también se han obtenido una gran cantidad de resultados para diversas familias de polígonos y diversas fuentes de iluminación. Una *fuente de iluminación* es un guardia (o foco), un reflector, un módem o un θ_k-módem agregado en el interior de un polígono. Por estas razones surge la necesidad de tener una herramienta que ayude a visualizar el comportamiento de estos últimos en el interior de un polígono.

El desarrollo de estas herramientas se ha realizado principalmente para apoyar la obtención de resultados de alguna variante del problema en particular, dejando dichas herramientas un poco restringidas en cuanto a las opciones de uso, es decir, resulta complicado utilizar alguna de estas herramientas como apoyo en el estudio de alguna otra variante del problema. Por esta razón se desarrolló la aplicación *LitPolygons*, esta aplicación permite dibujar polígonos de diferentes tipos de forma interactiva y dinámica, esto con el fin de dar al usuario flexibilidad y control sobre el diseño de éstos. La tarea principal de esta aplicación es calcular el polígono de iluminación de diferentes fuentes de iluminación, la aplicación permite al usuario elegir el punto donde se ubicará dicha fuente; una vez ubicada la fuente, ésta se puede mover, ocultar o eliminar a decisión del usuario.

El avance en desarrollo tecnológico que se tiene hoy en día ha provocado el auge en el uso de dispositivos móviles como herramientas de uso cotidiano. Tomando en cuenta esto, *LitPolygons* se implementó con la opción de ser ejecutada en dispositivos móviles, esta característica merece resaltarse puesto que hasta la fecha, las aplicaciones utilizadas en el área de la geometría computacional que se pueden usar en un dispositivo móvil son escasas y en algunos casos inexistentes. Ejemplo de este tipo de aplicaciones es *GeoGebra*,

que realiza tareas genéricas tales como dibujar puntos, rectas, polígonos, así como calcular intersecciones y distancias entre ellos.

6.2. Objetivos

- Dibujar polígonos de diferentes tipos: simples, monótonos, ortogonales y ortogonales-monótonos.
- Permitir mover vértices y conservar las propiedades de cada tipo de polígono.
- Calcular el polígono de visibilidad de un foco, un módem o un θ_k-módem.
- Mover fuentes de luz y recalcular el polígono de visibilidad.
- Permitir mover vértices y recalcular el polígono de visibilidad.
- Guardar la imagen en formato *PNG* o *SVG*.
- Abrir imagen *SVG*, previamente creada con la aplicación, para su edición.

6.3. Descripción de la aplicación

LitPolygons es una aplicación geométrica que ayuda a visualizar de forma interactiva cómo un conjunto de focos, módems o θ_k-módems iluminan el interior de un polígono. Las dos principales tareas que realiza la aplicación son: dibujar y editar de forma dinámica un polígono y agregar fuentes de iluminación al interior de éste. Para esto se realizó la implementación de varios de los algoritmos mostrados en este trabajo. Específicamente, para realizar la primer tarea se implementaron los algoritmos para la construcción de polígonos descritos en el capítulo 3. Para realizar la segunda tarea se implementó el algoritmo *PoligonoK-visibilidad* junto con las funciones necesarias, descritas en la sección 5.2.2.

La aplicación *LitPolygons* realiza también las tareas de guardar en una imagen *SVG* o *PNG* los elementos dibujados, así cómo abrir los archivos *SVG* creados con la aplicación para su edición, permitiendo mover los vértices del polígono, agregar, eliminar o mover las fuentes de iluminación. Es importante recalcar que los archivos del tipo *SVG* que se pueden abrir con la aplicación son únicamente aquellos que fuerón creados con la misma. El formato *SVG* almacena los datos de los elementos que contiene, la forma de almacenarlos es mediante etiquetas, diversas etiquetas permiten realizar tareas similares, por lo que no existe una forma única de almacenar datos usando este formato y crear una aplicación que pueda abrir de manera eficiente las distintas formas en que se pueden almacenar los mismos datos en un archivo *SVG* no es prioridad en el desarrollo de este trabajo.

6.3.1. Elementos de la aplicación

La aplicación consta de una interfaz gráfica de usuario, ver figura 6.1, dicha interfaz contiene los siguientes elementos:

Figura 6.1: Interfaz gráfica de usuario

1. **Menú principal:** Este menú consta de tres opciones: *Archivo*, *Polígono* y *Limpiar*. El menú *Archivo* contiene la opción *abrir*, la cual permite abrir un archivo *SVG*. También contiene las opciones de *guardar SVG* y *guardar PNG*, que permite guardar una imagen del *Área de dibujo* en formato PNG y SVG respectivamente. El menú *Polígono* consta de cuatro opciones: *Simple*, *Monótono*, *Ortogonal* y *Ortogonal-Monótono*, cada opción permite al usuario dibujar un polígono del tipo seleccionado. El botón *Limpiar* elimina todos los objetos agregados a el *Área de dibujo*.

2. **Área de dibujo:** Este elemento es un lienzo en el cual el usuario puede dibujar el polígono que desea, también puede agregar las fuentes de iluminación que desee en la ubicación que el elija.

3. **Menú fuentes de iluminación:** Este menú consta de tres botones, el botón de *Agregar foco*, el botón de *Agregar módem* y el botón *Agregar reflector*. Como su nombre lo dice estos botones permiten agregar focos, módems y reflectores respectivamente al *Área de dibujo* así como calcular el polígono de visibilidad de éstos.

4. **Lista de fuentes de iluminación:** Al momento de agregar una fuente de ilumination al *Área de dibujo* se agrega un conjunto de botones a esta lista, si se agrega un foco o un módem se agregan dos botones, el primer botón permite ocultar y redibujar el polígono de visibilidad del foco o módem correspondiente y el segundo elimina el foco o módem del *Área de dibujo*. Si se agrega un reflector o un θ_k-módem se agregan cuatro botones, el primero y el cuarto realizan las tareas de ocultar y eliminar

respectivamente la fuente correspondiente, el segundo y tercero rotan en sentido antihorario y en sentido horario, respectivamente, sobre el punto que se ubica la fuente de iluminación el ángulo de visibilidad la fuente de iluminación correspondiente.

5. **Barra de estado:** Este elemento consta de cuatro etiquetas posicionadas una tras otra de forma horizontal, que despliegan información del *Área de dibujo*, La primera muestra el tipo de polígono que se está utilizando, la segunda muestra la cantidad de vértices agregados al dibujar un polígono. la tercera solo muestra el nombre de la aplicación, por último, al seleccionar una fuente de ilumación previamente agregada, la cuarta etiqueta muestra el tamaño del ángulo de visibilidad y la cantidad de paredes que puede atravesar dicha fuente.

6.4. Ejemplos de uso

6.4.1. Dibujando polígonos

Como ya se ha dicho *LitPolygons* cuenta con un *Área de dibujo* donde el usuario puede crear un polígono de los tipos disponibles en el menú *Polígono*. Por simplicidad el tipo de polígono que se construirá en esta sección sera un polígono simple.
Para dibujar nuestro polígono se utiliza el algoritmo 3, nombrado *poligonoSimple*, descrito en la sección 3.1. La forma de dibujar un polígono es muy sencilla, solo se tienen que agregar los vértices uno a uno haciendo click sobre el punto en el *Área de dibujo* donde se quiere ubicar el vértice, las aristas se forman uniendo consecutivamente los vértices agregados. Si un vértice no cumple con las propiedades del tipo de polígono este vértice no se agregará. Ver figura 6.2. Para ayudar en la tarea de dibujar, al hacer click sobre el área de dibujo, se muestra un par de líneas perpendiculares intersectadas en el punto donde se hizo click y paralelas a los ejes coordenados, que sirven de *guías* en la ubicación del nuevo vértice.

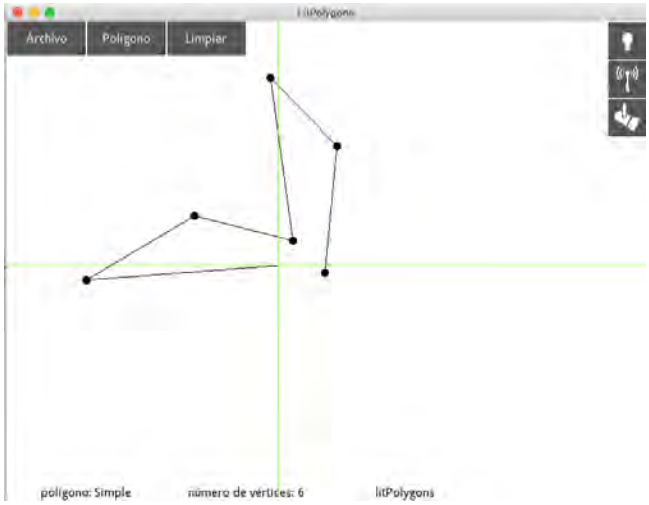
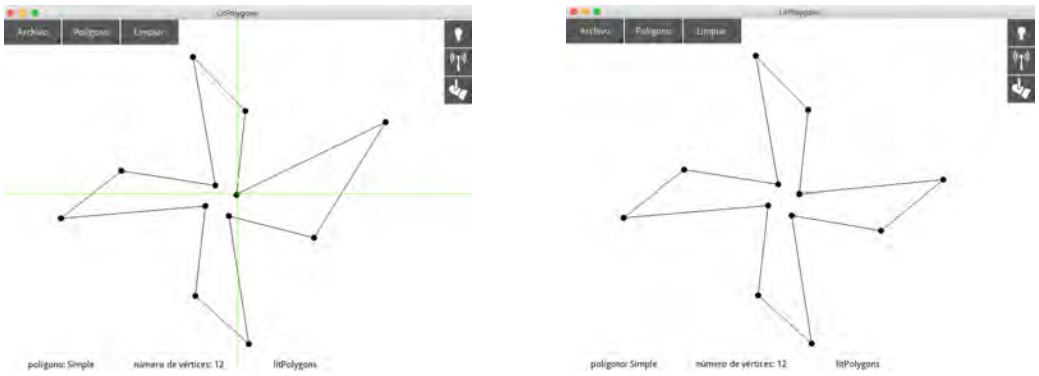


Figura 6.2: Ejemplo dibujar polígono

Para terminar de dibujar el polígono se tiene que cerrar la secuencia de aristas, es decir unir el último y el primer vértice. Los vértices agregados se pueden mover de posición si su ubicación final cumple con las propiedades del tipo de polígono, en caso contrario el vértice no se moverá. Ver figura 6.3.



(a) (b)

Figura 6.3: (a)Cerrar polígono (b)Mover vértice

6.4.2. Agregar fuentes de iluminación

Una vez terminado el polígono podemos agregar en su interior fuentes de iluminación. Para agregar un foco (o guardia) se tiene que activar el botón *Agregar foco* y dar click sobre el punto que se desea ubicar el foco, al finalizar el click se dibuja de un color elegido aleatoriamente el foco y el polígono de visibilidad del mismo. Ver figura 6.4.

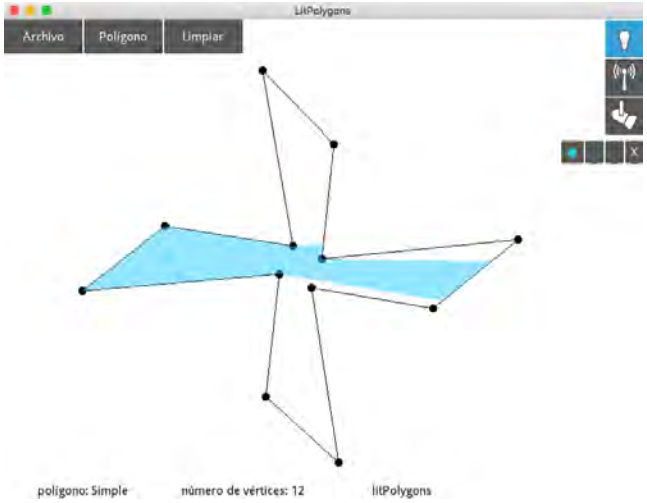


Figura 6.4: Ejemplo agregar foco

Como ya se ha mencionado *LitPolygons* permite mover la ubicación del foco, para esto se tiene que seleccionar el foco y arrastrarlo hasta su nueva ubicación, al soltar el foco se recalcula el polígono de visibilidad de éste. Ver figura 6.5.

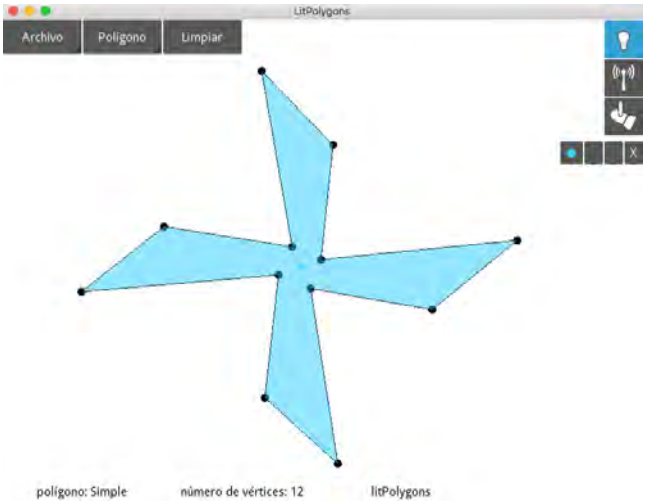


Figura 6.5: Ejemplo mover foco

La forma de agregar un módem es análoga, se activa el botón *Agregar módem* y se elige el punto sobre el que se desea ubicar el módem, al finalizar el click se muestra una ventana emergente donde se debe especificar la cantidad de paredes que le permitiremos atravesar, a dicho valor le llamamos el *poder* del módem. Ver figura 6.6.

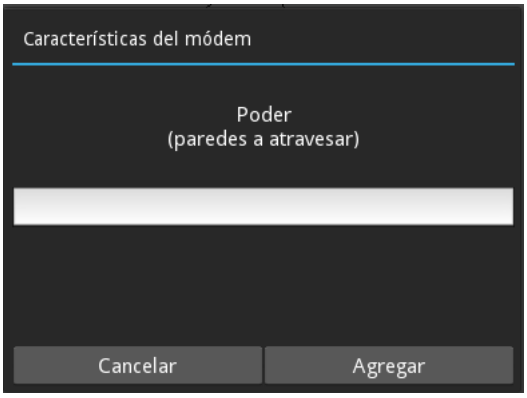


Figura 6.6: Ventana características del módem

Por último al hacer click en el botón *Agregar* se dibuja el módem y su polígono de visibilidad. Ver figura 6.7.

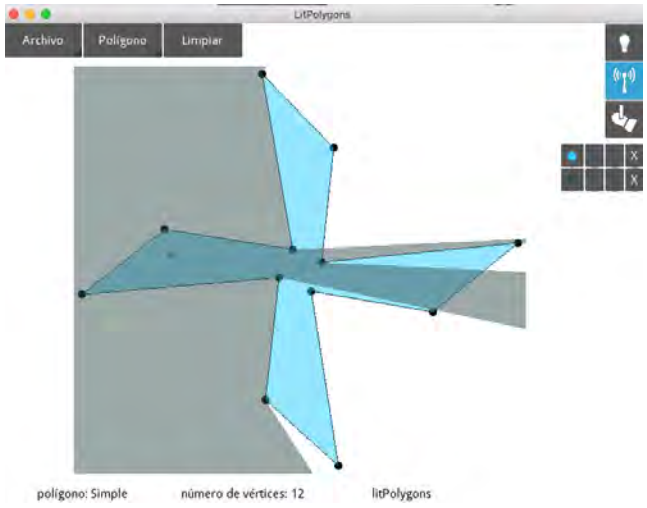


Figura 6.7: Ejemplo agregar un módem de poder 1

De igual forma para agregar un reflector o un θ_k-*módem* es activar el botón *Agregar reflector* y elegir el punto en el interior del polígono donde se ubicará la fuente de iluminación, esto muestra una ventana emergente donde se deben ingresar las características de ésta: el *poder* y el ángulo de visibilidad. Ver figura 6.8. Si deseamos agregar un reflector el poder de nuestra fuente debe ser igual a cero.

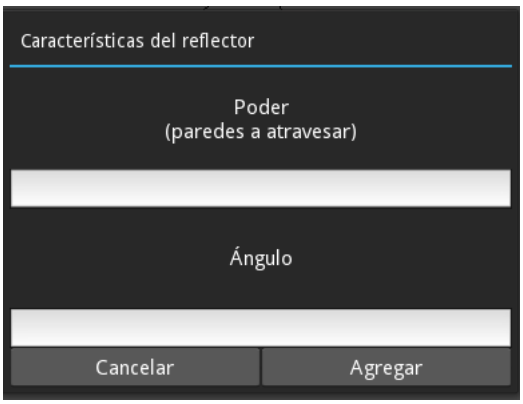


Figura 6.8: Ventana características del reflector

Al hacer click sobre el botón *Agregar* se dibuja el reflector o θ_k-módem especificado y su polígono de visibilidad. Ver figura 6.9.

Al agregar una fuente de iluminación se agregan a la *Lista de fuentes de iluminación* los botones que permiten ocultar el polígono de visibilidad y eliminar la fuente correspondiente. En el caso de los reflectores se agregan también los botones para rotar su ángulo de visibilidad. Si se activa el primer botón se oculta el polígono de visibilidad de la fuente

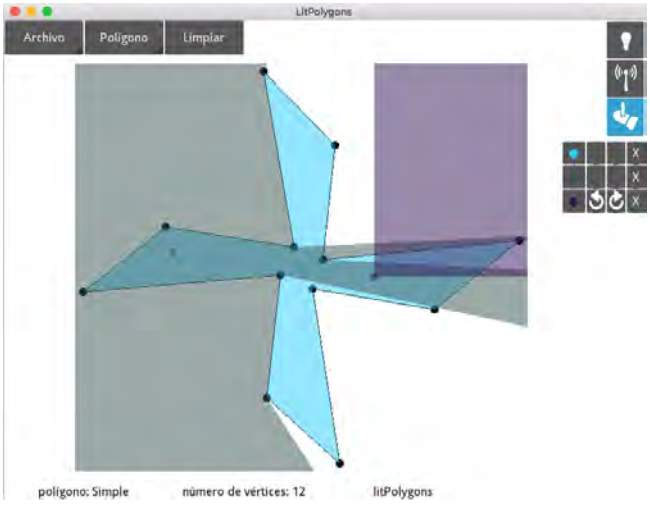


Figura 6.9: Ejemplo agregar reflector de poder 1 y ángulo de 90 grados

correspondiente, al desactivar este botón se vuelve a mostrar el polígono de visibilidad. Ver figura 6.10.

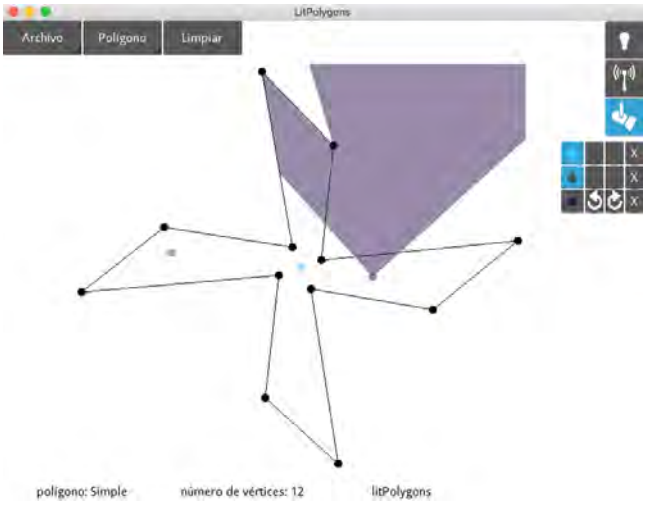


Figura 6.10: Ejemplo ocultar fuentes y rotar reflector

6.4.3. Guardar imagen

En cualquier momento durante el uso de la aplicación se puede guardar el estado del *Área de dibujo* en una imagen en formato PNG o SVG. Esta tarea es muy genérica y sigue la filosofía de la mayoría de las aplicaciones. Del menú *Archivo* se elige la opción *Guardar PNG* o *Guardar SVG*, lo que abre la ventana *Guardar imagen*, se elige la ubicación donde

se guardará la imagen y se le asigna un nombre a la imagen. Por último se hace click sobre el botón de *Guardar*. Ver figura 6.11.

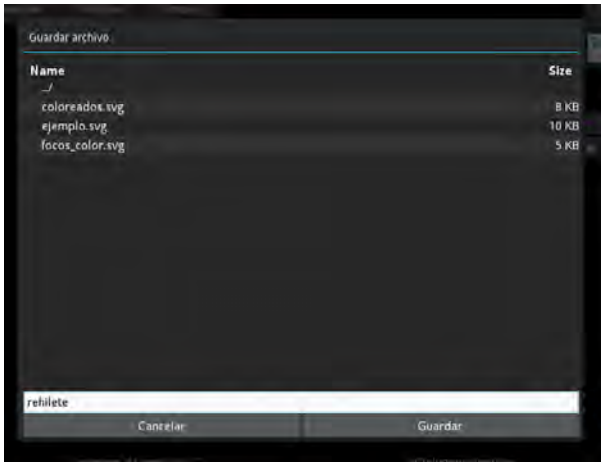
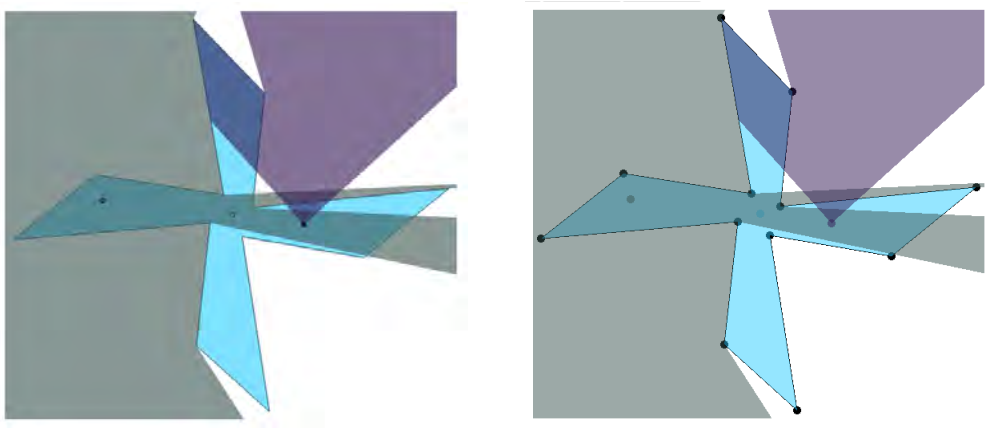


Figura 6.11: Ejemplo guardar imagen

Las siguientes imágenes fueron creadas con la aplicación *LitPolygons*:



(a) (b)

Figura 6.12: (a)Ejemplo imagen en formato SVG (b)Ejemplo imagen en formato PNG

6.4.4. Abrir SVG

Para abrir una imagen SVG, previamente construida con *LitPolygons*, del menú *Archivo* se elige la opción *Abrir SVG* lo que abre la ventana *Abrir imagen*, se elige el archivo y se hace click sobre el botón *Abrir*. Ver figura 6.13.

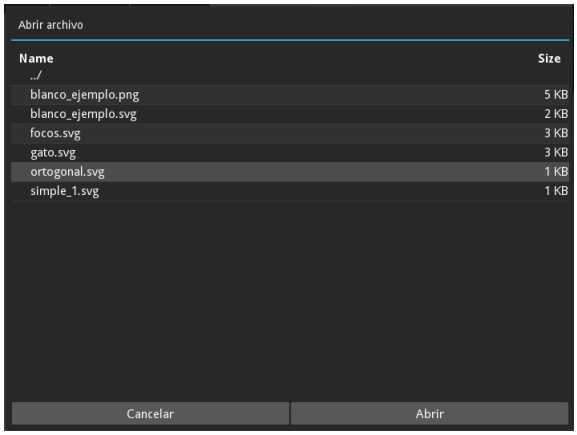


Figura 6.13: Ejemplo abrir imagen SVG

6.5. Requerimientos y tecnologías

En el desarrollo de la aplicación *LitPolygons* se utilizarón las siguientes tecnologías:

- **Lenguaje de programación:** Python 2.7
- **Framework:** Kivy 1.9
- **Bibliotecas adicionales:** SVGWrite 1.1.9
- **Control de versiones:** Git 2.10

Se eligió el lenguaje de programación Python por su simplicidad de sintaxis, su transparencia y fácil legibilidad de código, todos los algoritmos implementados para uso de la aplicación se desarrollaron en este lenguaje. Ya establecido el lenguaje de programación a usar, se buscó la herramienta que ayudó en la tarea de visualizar el resultado de estos algoritmos, para esto se eligió el marco de trabajo Kivy, Kivy es una herramienta de *código abierto*[1] de Python que permite el desarrollo simple de aplicaciones que usan interfaces de usuario con soporte multi-touch para dispositivos móviles. En la aplicación *LitPolygons*, Kivy se utilizó para desarrollar la interfaz de usuario. Esta herramienta se puede usar como una biblioteca de Python, como un lenguaje distinto (lenguaje Kivy) o una combinación de ambos [33, 42].
La biblioteca SVGWrite se utilizó como un auxiliar para crear imágenes en formato SVG a partir de los objetos agregados al área de dibujo de la aplicación. La elección de Git se basa en la necesidad de tener un control de versiones durante el desarrollo de la aplicación y su facilidad para el desarrollo de código basado en el *flujo de trabajo ramificado*[2], específicamente en el uso de *ramas puntuales*. Este tipo de ramas se utiliza comúnmente

[1]Descripción de código abierto (open source): https://opensource.org/osd
[2]Descripción de flujo de trabajo ramificado: https://git-scm.com/book/es/v1/Ramificaciones-en-Git-Flujos-de-trabajo-ramificados

en el desarrollo de funcionalidades concretas para ser eliminadas después de fusionarlas con la rama principal.

6.5.1. Repositorio

La aplicación se encuentra en el repositorio de código bitbucket.org, este repositorio utiliza Git como control de versiones y permite la creación de repositorios privados para grupos pequeños de desarrollo, dichas características motivaron la elección de este repositorio. El proyecto Git nombrado *litpolygons*, se puede encontrar en la dirección:

- `https://bitbucket.org/riclopez/litpolygons`

Documentación

La *Google Python Style Guide*[3] (Guía de Estilo Google Python), contiene una lista de reglas a seguir para el desarrollo de proyectos en el lenguaje de programación Python, utilizado por algunos grupos de desarrollo de Google.
Para la elaboración de la documentación del proyecto *litpolygons* se tomó un subconjunto de reglas de la *Google Python Style Guide*, esto con el fin de generar un repositorio de código legible y facilitar el uso de éste a otros desarrolladores.
La documentación del proyecto se puede encontrar en el repositorio arriba mencionado.

6.5.2. Instalación

La aplicación está disponible para dispositivos Android y se puede ejecutar en equipos Mac y la distrubución de Linux: Ubuntu.

Android

La extensión *APK*, es el formato de archivo utilizado para instalar programas en Android [33]. Para facilitar la instalación en dispositivos Android se construyó el empaquetado litpolygons.apk. Debido a que la aplicación no se encuentra en el catálogo de aplicaciones de la tienda oficial de Android, antes de iniciar con la instalación se debe acceder a la configuración del dispositivo y dar permiso de instalar aplicaciones de terceros. Activados estos permisos, la instalación de *LitPolygons* en un dispositivo Android se puede hacer siguiendo los siguientes pasos:

1. Copiar el archivo litpolygons.apk al dispositivo móvil.
2. Desde un navegador de archivos ubicar litpolygons.apk.
3. Hacer doble click sobre el archivo.
4. Dar permiso a la aplicación para acceder a los archivos del dispositivo.

[3]Guia de Estilo Google Python: https://google.github.io/styleguide/pyguide.html

Al finalizar la instalación, la aplicación se encuentra en la lista de aplicaciones del dispositivo. Si se desea acceder de manera más rápida, se puede agregar la aplicación a alguno de los escritorios del dispositivo, para esto solo se tiene que arrastrar de la lista de aplicaciones hacia el escritorio que se desee.

Mac OS

Los archivos con extensión *dmg*, son una *imagen de disco* comúnmente utilizada por el sistema operativo mac OS, que encapsulan aplicaciones, para su instalación fácil y rápida. Por lo que se ha creado la imagen litpolygons.dmg, la cual contiene la aplicación *LitPolygons*. Para la instalación de la aplicación en equipos con dicho sistema operativo, se debe tener instalado Python 2.7.
Puesto que *LitPolygons* es un proyecto de software libre ajeno a la tienda oficial de Apple, antes instalarla, se debe acceder a las *preferencias del sistema*, abrir la opción *seguridad y privacidad* y elegir la opción *cualquier sitio* de los permisos para instalar aplicaciones. La instalación de *LitPolygons* se puede realizar siguiendo los siguientes pasos:

1. Extraer la imagen litpolygons.dmg, haciendo doble click sobre ella. Esto montará una unidad de disco en el navegador de archivos, con la que se puede acceder al contenido de la imagen de disco.

2. Acceder a la unidad de disco recién montada y arrastrar la aplicación *LitPolygons* hacia la carpeta de Aplicaciones.

Para acceder a la aplicación, se puede utilizar la herramienta de busqueda *Spotlight*, escribir LitPolygons en dicho buscador y dar doble click sobre el icono de la aplicación.

Linux(Ubuntu)

LitPolygons se puede ejecutar en equipos con la distribución de linux Ubuntu. Para esto se debe tener instalado Python 2.7 y el marco de trabajo Kivy, lo cual se explica en el sitio oficial `kivy.org`. Los pasos a seguir para la ejecución de la aplicación son los siguientes:

1. Obtener el proyecto LitPolygons.

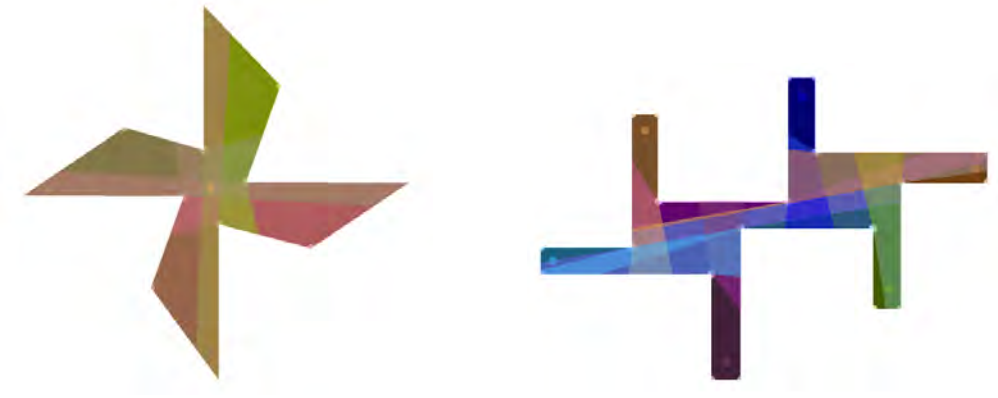
2. Desde una terminal ingresar al directorio raíz del proyecto.

3. Ejecutar:

    - `python main.py`

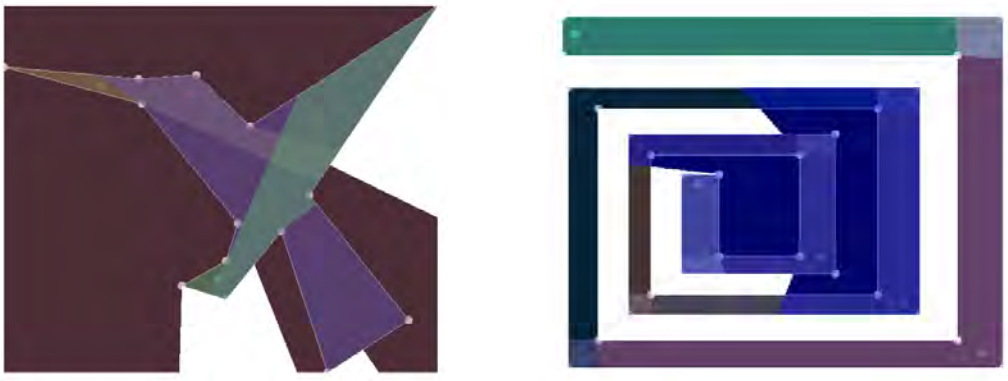
6.5.3. Galería de imágenes

A continuación se muestran algunos polígonos iluminados utilizando la aplicación *LitPolygons*.



(a) (b)

Figura 6.14: (a)Rehilete (b)Hélice



(a) (b)

Figura 6.15: (a)Colibrí (b)Espiral ortogonal

Capítulo 7

Conclusiones

Existe una gran variedad de problemas de interés para este trabajo sobre vigilar o iluminar una galería de arte, en particular sobre el problema de iluminación con módems. En los capítulos 4 y 5 se presentaron algunos de estos resultados elegidos principalmente por las técnicas utilizadas en su obtención.
Sobre los resultados obtenidos en este trabajo, se tiene lo siguiente:

- En el capítulo 3 se diseñó una serie de algoritmos para construir polígonos *simples, monótonos, ortogonales* y *ortogonales monótonos* de forma interactiva. Estos algoritmos fueron implementados para su uso en la aplicación *LitPolygons*. También para su uso en dicha aplicación se implementó el algoritmo 15: *PolígonoK-visibilidad*. El resultado final del trabajo de programación es la aplicación geométrica e interactiva *LitPolygons*, disponible para su instalación en dispositivos móviles Android y ejecutable en equipos de cómputo con sistemas operativos Mac y Ubuntu.

- En términos técnicos, un módem es una antena que transmite datos de forma inalámbrica. En la *realidad* existen dos tipos de antenas [39]:

  1. *Antena omnidireccional.* El patrón de transmisión es de 360 grados en el plano horizontal y de aproximadamente 75 grados en el plano vertical.
  2. *Antena direccional.* El patrón de transmisión es acotado por un ángulo en el plano horizontal y en el plano vertical.

  Si se tuviera que utilizar antenas direccionales para dar servicio de red inálambrica a un edificio, entonces este problema se pude representar por el problema de iluminación con θ_k-módems. A partir de este problema se puede obtener una nueva línea de investigación con posibles aplicaciones prácticas.

- El algoritmo *PolígonoK-visibilidad*, calcula el polígono de visibilidad de un k-módem, para lo cual utiliza un *barrido de línea angular*. Este algoritmo se puede modificar para calcular el polígono de visibilidad de un θ_k-módem acotando el barrido de línea por el ángulo de visibilidad θ, esto es, sean r_1 y r_2 las rectas que determinan θ, el barrido de línea comienza en r_1, procesa los vértices en el interior de θ y termina en

r_2. Al igual que *PoligonoK-visibilidad*, la complejidad del algoritmo es de $O(n \log n)$, pues en el peor de los casos todos los vértices del polígono se encuentran en el interior de θ.

- El algoritmo *PoligonoK-visbilidad* utiliza un rectángulo R que contiene al polígono para hacer discreta el área de iluminación de un k-módem, si k es mayor a la cantidad de aristas que bloquean la visibilidad del k-módem, entonces el área de iluminación se acota por R. En el barrido de línea angular se *salta* del polígono a R y viceversa las veces que sea necesario, Esta característica se puede aprovechar para modificar el algoritmo y calcular el polígono de visibilidad de un k-módem en el interior de un conjunto de polígonos anidados.

7.1. Trabajo a futuro

Durante el desarrollo de este trabajo se revisaron algunos temas que resultan interesantes, en los cuales se puede profundizar en su estudio:

- Seguir la línea de investigación del problema de iluminación con θ_k-módems.
- Diseñar un algoritmo eficiente que calcule el polígono de visibilidad de un k-módem dentro de un polígono con hoyos.
- Otra variante del problema de la galería de arte es el problema de vigilar la galería con *guardias arista* [3]. Extender el concepto de k-visibilidad de un guardia arista, es otra variante en la que se puede iniciar una nueva línea de investigación.
- Con fines más prácticos, los módems emiten su señal en tres dimensiones, por lo que extender el concepto de k-visibilidad a objetos en tres dimensiones, puede considerarse naturalmente el siguiente paso.

Bibliografía

[1] J. Abello, V. Estivill-Castro, T. Shermer, and J. Urrutia. Illumination of orthogonal polygons with orthogonal floodlights. *International Journal of Computational Geometry & Applications*, 08(01):25–38, 1998.

[2] O. Aichholzer, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, J. Urrutia, and B. Vogtenhuber. Modem illumination of monotone polygons. *Computational Geometry*, 2017.

[3] D. Avis and G. T. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Trans. Comput.*, 30(12):910–914, Dec. 1981.

[4] A. Bajuelos, S. Canales, G. Hernández, and A. Martins. Aproximando la iluminación por módems. In *XIII Spanish Workshop on Computational Geometry*, pages 67–74, 2009.

[5] B. Ballinger, N. Benbernou, P. Bose, M. Damian, E. D. Demaine, V. Dujmović, R. Flatland, F. Hurtado, J. Iacono, A. Lubiw, P. Morin, V. Sacristán, D. Souvaine, and R. Uehara. *Coverage with k-Transmitters in the Presence of Obstacles*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[6] I. Bjorling-Sachs and D. Souvaine. *A Tight Bound for Guarding General Polygons with Holes*. LCSR-TR-. Department of Computer Science, Rutgers University, 1991.

[7] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. North-Holland, New York, 1976.

[8] P. Bose, L. Guibas, A. Lubiw, M. Overmars, D. Souvaine, and J. Urrutia. The floodlight problem. *Proceedings of the Fifth Canadian Conference in Computational Geometry*, 1994.

[9] V. Brumberg, S. Ramaswami, and D. Souvaine. Experimental results on upper bounds for vertex pi-lights. In *Abstr. 11th Ann. Fall Workshop Comput. Geom.(Brooklyn, NY*, 2001.

[10] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[11] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees. *SIGGRAPH Comput. Graph.*, 23(3):99–106, jul 1989.

[12] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39 – 41, 1975.

[13] F. Contreras, J. Czyzowicz, E. Rivera-Campo, and J. Urrutia. Optimal floodlight illumination of stages. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 409–410, New York, NY, USA, 1998. ACM.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambrige, Masachusetts Londres, Inglaterra, 2009.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2008.

[16] F. Duque and C. Hidalgo-Toscano. An upper bound on the k-modem illumination problem. *International Journal of Computational Geometry & Applications*, 25(04):299–308, 2015.

[17] V. Estivil-Castro and J. Urrutia. Two-floodlight illumination of convex polygons. pages 62–73, 1995.

[18] V. Estivill-Castro, J. O'Rourke, J. Urrutia, and D. Xu. Illumination of polygons with vertex lights. *Information Processing Letters*, 56(1):9 – 13, 1995.

[19] V. Estivill-Castro and J. Urrutia. Optimal floodlight illumination of orthogonal art galleries. In *CCCG*, pages 81–86, 1994.

[20] R. Fabila-Monroy, A. R. Vargas, and J. Urrutia. On modem illumination problems. *XIII encuentros de geometría computacional, Zaragoza, Spain*, 2009.

[21] S. Fisk. A short proof of chvátal's watchman theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374 –, 1978.

[22] R. Fulek, A. F. Holmsen, and J. Pach. Intersecting convex sets by rays. *Discrete & Computational Geometry*, 42(3):343–358, 2009.

[23] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175 – 179, 1978.

[24] F. Hoffman, M. Kaufmann, and K. Kriegel. The art gallery theorem for polygons with holes. *Procedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 39 – 48, 1991.

[25] J. Kahn, M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *SIAM Journal on Matrix Analysis and Applications*, 4(2):194–13, 06 1983.

[26] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[27] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inf. Theor.*, 32(2):276–282, Mar. 1986.

[28] A. Lubiw. Decomposing polygonal regions into convex quadrilaterals. pages 97–106, 1985.

[29] J. O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, New York, USA, 1987.

[30] J. O'Rourke. *Computational geometry in C*. Cambrige University Press, Cambrige, Reino Unido, 1997.

[31] J. O'Rourke. Vertex pi-lights for monotone mountains. In *Proc. 9th Canadian Conference on Computational Geometry*, pages 1–5, Ontario, Canada, 1997.

[32] J. O'Rourke, T. Shermer, and I. Streinu. Illuminating convex polygons with vertex floodlights. *Seventh Canadian Conference in Computational Geometry*, pages 151 – 156, 1995.

[33] D. Phillips. *Creating Apps in Kivy*. O'Reilly Media, USA, 2014.

[34] L. Ricci, L. Genovali, and B. Guidi. *Managing Virtual Entities in MMOGs: A Voronoi-Based Approach*, pages 58–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[35] D. T. Robert Laurini. *Fundamentals of Spatial Information Systems*. Academic Press, 1992.

[36] J.-R. Sack and J. Urrutia. *Handbook of Computational Geometry*. Elsevier, Amsterdam, The Netherlands, 2000.

[37] B. Speckmann and C. D. Tóth. Allocating vertex π-guards in simple polygons via pseudo-triangulations. *Discrete & Computational Geometry*, 33(2):345–364, 2005.

[38] R. E. Tarjan and C. J. V. Wyk. An $o(n \log \log n)$-time algorithm for triangulating a simple polygon. *SIAM, Journal of Computing*, 17:143 – 178, 1988.

[39] ©Cisco Systems. Omni antena vs directional antena. http://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/82068-omni-vs-direct.html, 2007. [Web; accedido el 20-12-2016].

[40] C. D. Tóth. *Illuminating Polygons with Vertex π-Floodlights*, pages 772–781. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[41] C. D. Tóth. Illumination of polygons by 45-floodlights. *Discrete Mathematics*, 265(1):251 – 260, 2003.

[42] R. Ulloa. *Kivy: Interactive Application in Python.* Packt Publishing Ltd, 2013.

[43] J. Urrutia. Iluminando polígonos con reflectores. In *Proc. IV Encuentro de Geometría Computacional*, pages 59–72, Barcelona, España, 1995.

[44] J. Urrutia. *Art Gallery and Illumination Problems.* México. D.F., México, 2004.

[45] M. F. Worboys. A generic model for planar geographical objects. *International Journal of Geographical Information Systems*, 6(5):353–372, 1992.