Instituto Politécnico Nacional

Escuela Superior de Computo

Geometría Computacional.

Proyecto final. Posicionamiento de cámaras para una óptima vigilancia

Profesora: Palma Orozco Rosaura

Alumno: Pimentel Paulin Daniel Jezrael

daniel.pim3ntel@gmail.com

Grupo 3CM16

Contenido

Lista de Figuras

Objetivo

El principal objetivo del presente proyecto es implementar un algoritmo que permita calcular el posicionamiento de cámaras para una optima vigilancia dentro de un departamento o casa.

Marco teórico

Tras una breve investigación, se observó que el principal objetivo de este trabajo puede verse como una solución al problema del museo.

El problema del museo o también llamado problema de la galería de arte es un problema de visibilidad estudiado en la geometría computacional. Fue propuesto por Victor Klee en 1973 y consiste en determinar el número mínimo de guardias necesarios para cubrir cada rincón de un museo. Solo que, para nuestro caso en particular, en lugar de considerar guardias consideraremos cámaras de seguridad.

Definiendo el problema, el museo será representado como un polígono simple de n vértices, y las cámaras de seguridad como puntos en el polígono.

Polígono simple

Un polígono simple es una región cerrada conectada cuyo limite está definido por un numero finito de segmentos de línea. De forma más estricta un polígono simple (P) se define como una secuencia de puntos $\mathrm{v}_0, \mathrm{v}_1, \ldots, \mathrm{v}_\mathrm{n} - 1, \mathrm{v}_0$ en el plano, y una secuencia correspondiente de segmentos de línea o aristas $\mathrm{v}_0\mathrm{v}_1, \mathrm{v}_1\mathrm{v}_2, \ldots, \mathrm{v}_{\mathrm{n}-2}\mathrm{v}_{\mathrm{n}-1}, \mathrm{v}_{\mathrm{n}-1}\mathrm{v}_0$, donde los bordes no consecutivos de la secuencia no se intersecan. Un polígono simple divide al plano que lo contiene en dos conjuntos de puntos: interior de la región poligonal y exterior de la región poligonal.

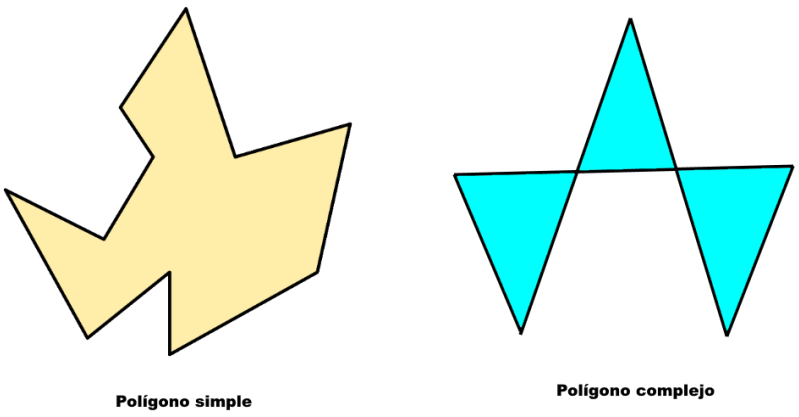

Figura 1. Ejemplo de polígono simple

Visibilidad

Dado un polígono simple P, se dice que dos puntos p y q son visibles entre sí, sí y solo si el segmento de línea pq no corta el exterior de P. Si p y q son mutuamente visibles y el segmento de línea pq se encuentra dentro de P entonces se dice que son internamente visibles entre sí. Por el contrario, si el segmento de línea pq se encuentra fuera de P, se dice que son visibles desde el exterior.

Numero de cámaras

Una vez mostradas estas definiciones, nos queda resolver la incógnita principal: cual es el número mínimo de cámaras de seguridad requeridas para vigilar todo un museo con n paredes. Para ello primero hay que definir lo que una cámara es capaz de observar: asumiremos que una cámara de seguridad es capaz de vigilar los 360 grados desde su posición haciendo una rotación.

Existe un teorema propuesto por Chvátal, (el cual fue demostrado por Steve Fisk utilizando el coloreado de grafos en 1978), el cual menciona que un museo puede ser vigilado a lo más por $n/3$ cámaras.

Teorema 1 (Teorema de la galería de arte).

Son suficientes n⁄3 cámaras de seguridad para vigilar un museo con n paredes (polígono de n vertices).

Demostración

Se probará este teorema a través de una secuencia de afirmaciones. Pero primero, se mostrarán algunos polígonos de ejemplo y el respectivo número de cámaras necesarias para asegurar que toda el área esté protegida. Luego se introducirá a la triangulación y el coloreado de vértices para encontrar el límite generalizado de $n/3$ cámaras.

Casos base

- Solo una cámara necesaria:

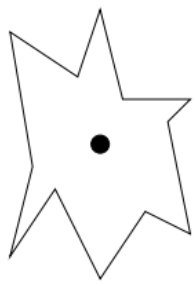

Figura 2. Estrella

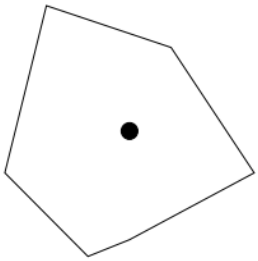

Figura 3. Polígono convexo

- Dos cámaras necesarias:

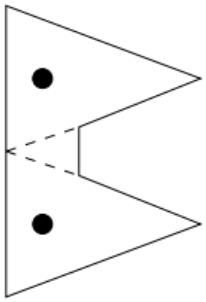

Figura 4. Polígono de 6 lados

Ahora se generalizará cómo determinar el número de guardias necesarios para cualquier polígono de n vértices. Primero es necesario triangular el polígono. La triangulación de un polígono es una descomposición del polígono en triángulos dibujando diagonales que no se intersecan entre pares de vértices.

Afirmación 1.

Cualquier polígono P puede ser triangulado

Demostración

Se probará esta afirmación por inducción sobre el número n de vértices. Para $n = 3$ el polígono P es un triángulo, que ya está triangulado. Para $\mathrm{n} >= 4$, encontraremos una sola diagonal (es decir, un segmento de línea que se encuentra dentro de P, que conecta un par de vértices), que divide el polígono en dos partes más pequeñas. Luego se puede obtener la triangulación de todo el polígono pegando la triangulación de las dos partes diferentes. Como la suma de los ángulos interiores de P es $(\mathrm{n} - 2) * 180°$, hay un vértice u de P con ángulo interior menor que $180°$. Sean v, w los vértices vecinos de u en el polígono. Si el segmento de línea v, w se encuentra dentro del polígono, entonces este segmento de línea es la diagonal deseada. De lo contrario, el triángulo $\triangle \mathrm{uvw}$ contiene otros vértices. Moviendo el segmento de línea v, w hacia u hasta que llegue al vértice final x en el triángulo $\triangle \mathrm{uvw}$. Entonces, el segmento de línea ux se encuentra dentro del polígono y se puede tomar como la diagonal deseada, lo que prueba la afirmación.

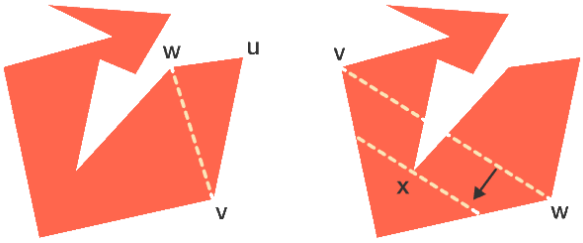

Figura 5. Proceso de triangulación de un polígono

Afirmación 2.

Cualquier polígono triangulado es tres colorable.

Demostración

Se probará por inducción sobre el número de vértices del polígono. Para $n = 3$, el polígono es un triángulo y podemos elegir tres colores diferentes para los tres vértices. Ahora, considerando cualquier polígono triangulado con $\mathrm{n} > 3$ vértices. Escogiendo dos vértices u y v que estén conectados por una diagonal (es decir, conectados por un borde en la triangulación, pero no en el polígono original) podemos dividir el polígono en dos polígonos triangulados usando el borde (u, v) y por inducción, los dos polígonos triangulados son de 3 colores.

Sean (rojo, azul, verde) los colores de la primera triangulación T_1 y sean (1,2,3) los colores de la segunda triangulación T_2. Identificamos el color de u en la primera triangulación con el número que etiqueta u en la segunda triangulación, e identificamos el color de v en la primera triangulación con el número que etiqueta v en la segunda triangulación. Luego, identificamos los últimos colores restantes en ambas triangulaciones entre sí.

Finalmente obtenemos una coloración para toda la triangulación conservando el color de todos los vértices en la primera triangulación y usando los colores identificados con (1,2,3) para los vértices en la segunda triangulación. Esto da una coloración en 3 de todo el polígono triangulado y prueba la Afirmación 2.

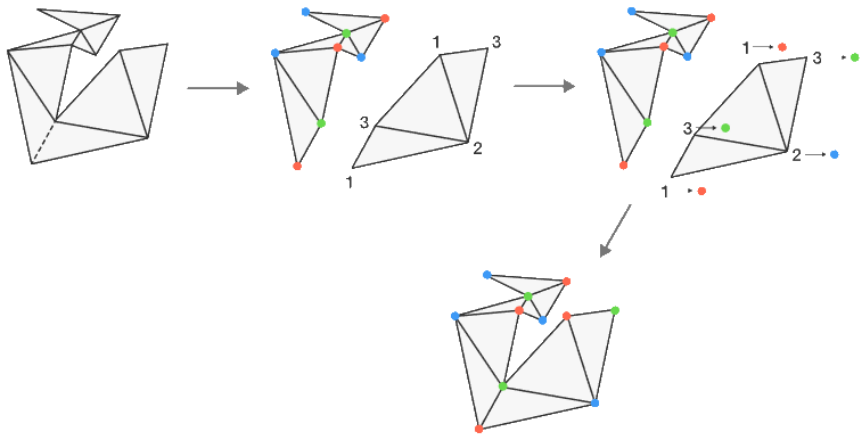

Figura 6. Proceso de coloración de un polígono

Afirmación 3.

Para cualquier coloración 3 de una triangulación, existe un color tal que el número de vértices de este color es $\leq \mathbf{n/3}$, y colocar cámaras en estos vértices protegerá todo el museo.

Demostración

Sin pérdida de generalidad, supongamos que los colores de los vértices son rojo, verde, azul tales que el número r de vértices rojos es menor o igual que el número g de vértices verdes, que es menor o igual que el número b de vértices azules. El número total de vértices es n, por lo que $r + g + b = n$. Si $r > n/3$ entonces g y b también serían estrictamente mayores que $n/3$, y la identidad $r + g + b = n$ no se puede satisfacer. Por lo tanto, $r <= n/3$. Ahora, al colocar una cámara en cada vértice coloreado en rojo, observe que cada triángulo en la triangulación tiene exactamente un nodo rojo y, por lo tanto, exactamente una cámara. Además, cualquier punto P en el museo está contenido en un triángulo en el polígono triangulado, y P es visible desde el vértice del triángulo coloreado en rojo. Esto da una ubicación de $n/3$ cámaras como máximo que vigilan todo el museo, lo que demuestra la Afirmación 3.

Las afirmaciones 1, 2 y 3 juntas dan una demostración del teorema de la galería de arte.

Triangulaciones

Ahora bien, Fisk en su demostración nos habla de triangulaciones para obtener el posicionamiento de las cámaras. Pero ¿cómo obtenemos dichas triangulaciones?

Existen una diversidad de algoritmos que nos permiten triangular un polígono. Para nuestro trabajo, se utilizará el "Método de recorte de orejas" (Ear clipping method). Este método se basa en el teorema de las dos orejas, el cual no demostraremos.

Teorema 2.

Cualquier polígono simple con al menos 4 vértices sin agujeros tiene al menos dos 'orejas', que son triángulos con dos lados siendo los bordes del polígono y el tercero completamente dentro de él.

El algoritmo consiste en encontrar una oreja, eliminarla del polígono (lo que da como resultado un nuevo polígono que aún cumple con las condiciones) y repetir hasta que solo quede un triángulo.

Este algoritmo es fácil de implementar, pero más lento que otros algoritmos. Una buena implementación requerirá de un tiempo $O(n^2)$. Este algoritmo fue descubierto por Hossam ElGindy, Hazel Everett y Godfried Toussaint.

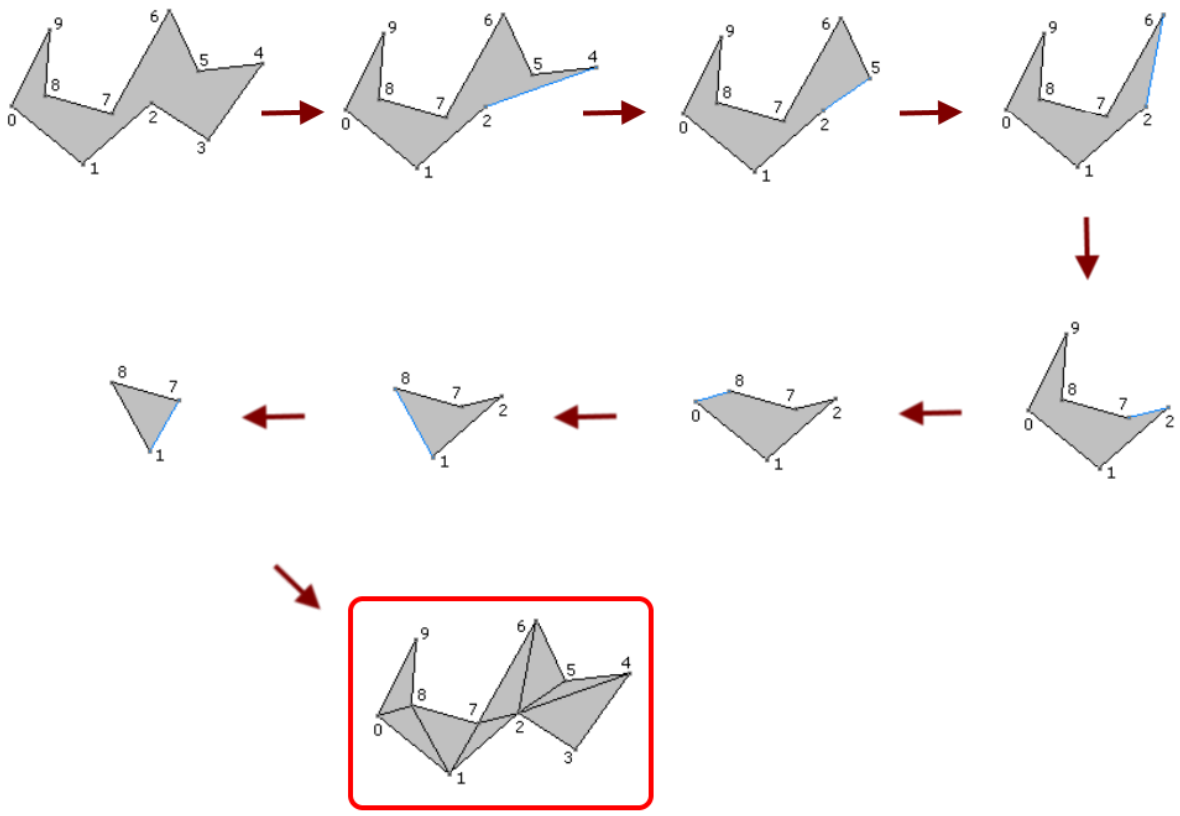

Figura 7. Triangulación utilizando el método de recorte de orejas

Material y equipo.

Para el desarrollo del proyecto, se utilizó el lenguaje de programación Python.

Python cuenta con una librería llamada Tripy la cual facilita la triangulación de polígonos. Utiliza el método de recorte de orejas mencionado en la introducción.

También se hizo uso de la librería Matplotlib.pyplot la cual permite un graficado muy similar a la aplicación MATLAB.

Desarrollo

Primero, se mostrará el algoritmo utilizado. Posteriormente se detallará lo que realiza cada una de las funciones del programa y al final se mostraran los resultados obtenidos.

Algoritmo

Para la búsqueda del posicionamiento de las cámaras se realizó lo siguiente:

1. Crear un polígono.
2. Triangular el polígono utilizando la librería Tripy.
3. Almacenar la lista de triángulos.
4. Buscar el vértice más frecuente en la lista de triángulos. (Aquí hay que tener en cuenta que existirá una lista negra la cual nos indicara que puntos ya no están disponibles)
5. Almacenar el vértice en la lista de las cámaras finales.
6. Actualizar la lista de triangulaciones borrando todas las triangulaciones que contengan el punto más frecuente.
7. Actualizar la lista negra añadiendo todos los puntos de las triangulaciones eliminadas.
8. Repetir desde el paso 4 hasta no encontrar un vértice más frecuente.
9. Graficar el polígono, la triangulación y las cámaras de seguridad

Funciones del programa

- Función most_frec()

La siguiente función obtiene el punto más frecuente en los triángulos. Es importante mencionar que se eliminan todos los puntos que se encuentran en la lista negra.

```
#Funcion que obtiene el punto más recurrente
def most_freq(poligon, triangles):
    global black_list_points
    max_count = 0
    list = []
    final_list = []
    most = False
    #Se separa cada punto de la lista de triángulos en una lista nueva
    for i in poligon:
        for j in triangles:
            for k in range(0,3):
                #print(j[k])
                if i == j[k]:
                    list.append(j[k])
```

```
    #Si existen puntos en la lista negra se eliminan de la lista
    for i in list:
        if not (i in black_list_points):
            final_list.append(i)

    #Se busca el punto más recurrente de la lista
    for i in final_list:
        current = final_list.count(i)
        if(current > max_count):
            max_count = current
            most = i
    return most
```

- Función remove_triangles ()

La siguiente función devuelve una lista actualizada. Se encarga de eliminar cada uno de los triángulos contenedores del punto más frecuente. Además, se agrega cada uno de estos puntos a la lista negra.

```
#Funcion que remueve todos los triángulos que contengan el punto mas
frecuente. Además agrega a la lista negra los puntos pertenecientes a dichos
triángulos
def remove_triangles(triangles, point):
    list = []
    black_list_triangles = []
    #Se eliminan los triángulos que contienen el punto más frecuente
    for i in triangles:
        if point in i:
            black_list_triangles.append(i)
        else:
            list.append(i)
    global black_list_points

    #Se agregan a la lista negra los puntos de dichos triángulos
    for i in black_list_triangles:
        for j in i:
            if j not in black_list_points:
```

```
                black_list_points.append(j)

    return list
```

- Función final_cameras ()

La siguiente función devuelve una lista con el posicionamiento de las cámaras de seguridad.

```
#Funcion que obtiene las cámaras a colocar
def final_cameras(polygon, triangles):
    cameras = []
    #Mientras existan triangulaciones
    while triangles != []:
        #Se busca el punto más frecuente de las triangulaciones
        m = most_freq(polygon, triangles)
        if not m:
            break

        #Se agrega el punto a la lista de cámaras
        #print("\nMost f : ", m)
        cameras.append(m)
        #Se eliminan las triangulaciones que contengan dicho punto
        triangles = remove_triangles(triangles, m)
        #print("Triangles: ", triangles)
    return cameras
```

- Función find_and_graph()

La siguiente función grafica el polígono, las triangulaciones y las cámaras obtenidas.

```
#Funcion que grafica el polígono, las triangulaciones y las cámaras
def find_and_graph(polygon):
    #print("Polygon: ", polygon)
    #Se obtiene triangulación
    triangles = tripy.earclip(polygon)
    #print("Triangles: ", triangles)

    #Listas de puntos utilizadas para graficar el polígono
```

```
plot_poligonx = [];  plot_poligony = []
for i in polygon:
    #Se obtiene las coordenadas en x del polígono
    plot_poligonx.append(i[0])
    #Se obtiene las coordenadas en y del polígono
    plot_poligony.append(i[1])

#Se agrega el primer punto de nuevo
plot_poligonx.append(plot_poligonx[0])
plot_poligony.append(plot_poligony[0])

#Listas de puntos utilizadas para graficar las triangulaciones
plot_trianglex = []; plot_triangley = []
index = 0
for i in triangles:
    aux_trianglex = []
    aux_triangley = []
    for j in i:
        #Se obtiene las coordenadas en x de la triangulación
        aux_trianglex.append(j[0])
        #Se obtiene las coordenadas en y de la triangulación
        aux_triangley.append(j[1])
    aux_trianglex.append(i[0][0])
    aux_triangley.append(i[0][1])

    plot_trianglex.append(aux_trianglex)
    plot_triangley.append(aux_triangley)

#Se obtiene la lista de cámaras
cameras = final_cameras(polygon, triangles)

#Listas de puntos utilizadas para graficar las cámaras
plot_camerax = []; plot_cameray = []
for i in cameras:
    plot_camerax.append(i[0])
    plot_cameray.append(i[1])

#Se grafican las camaras
plt.scatter(plot_camerax, plot_cameray, marker=".", s=500, color='k')
```

```
    #plt.plot(plot_trianglex, plot_triangley, color='g', alpha=0.3)

    #Se grafican las triangulaciones
    for i in range(len(plot_trianglex)):
        plt.plot(plot_trianglex[i], plot_triangley[i], color='g', alpha=0.1)

    #Se grafica el polígono
    plt.plot(plot_poligonx, plot_poligony, color='b')

    return plt.show()
```

Resultados obtenidos

Se realizaron varias pruebas con distintos polígonos.

- Prueba 1. Polígono de 4 lados (cuadrado)
  Se obtuvo un resultado de 1 cámara

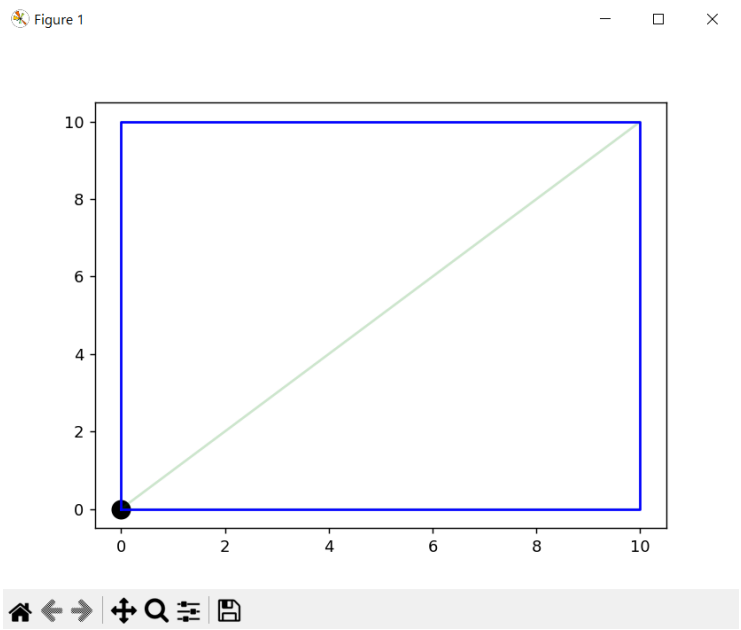

Figura 8. Resultado obtenido de la prueba 1

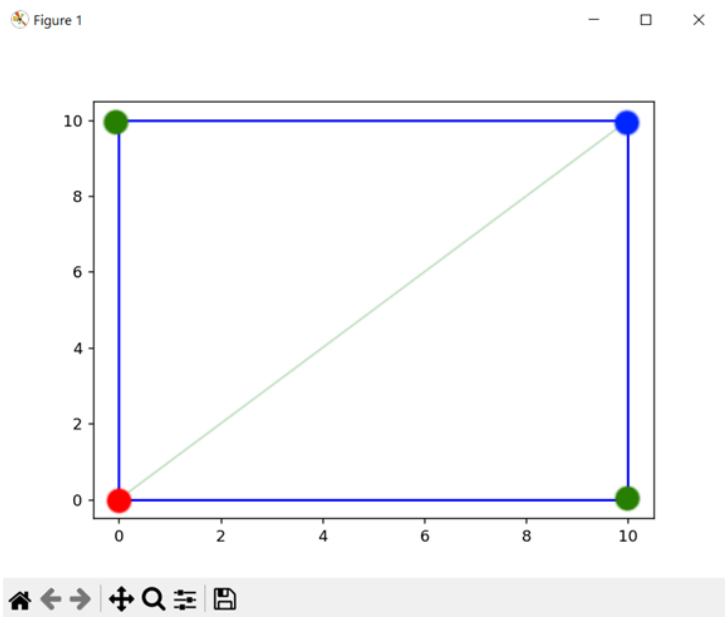

Figura 9. Coloración-3 de las triangulaciones prueba 1

- Prueba 2. Polígono de 11 lados
  Se obtuvo un resultado de 3 cámaras

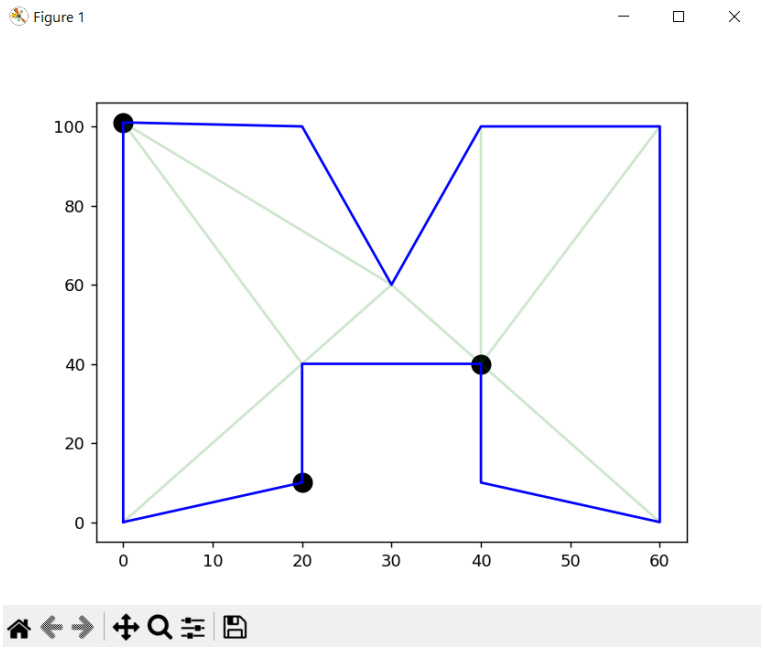

Figura 10. Resultado obtenido de la prueba 2

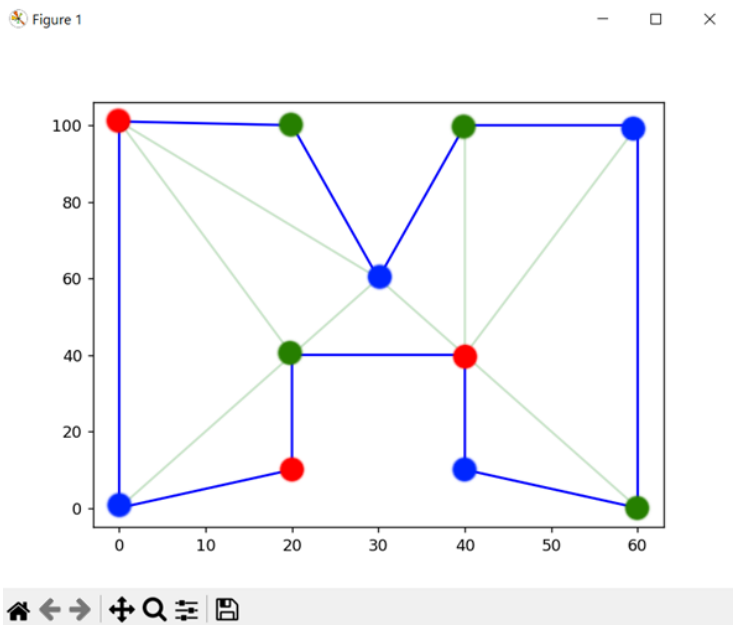

Figura 11. Coloración-3 de las triangulaciones prueba 2

- Prueba 3. Polígono de 20 lados
  Se obtuvo un resultado de 6 cámaras

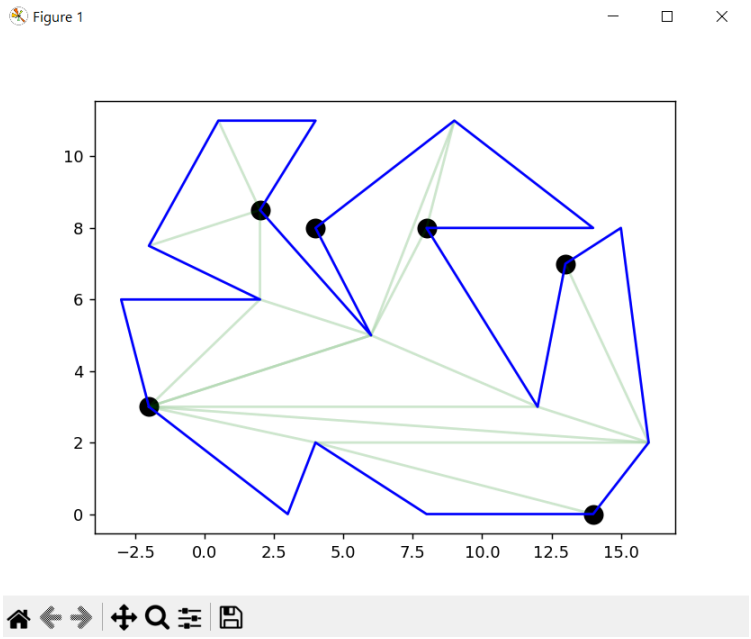

Figura 12. Resultado obtenido de la prueba 3

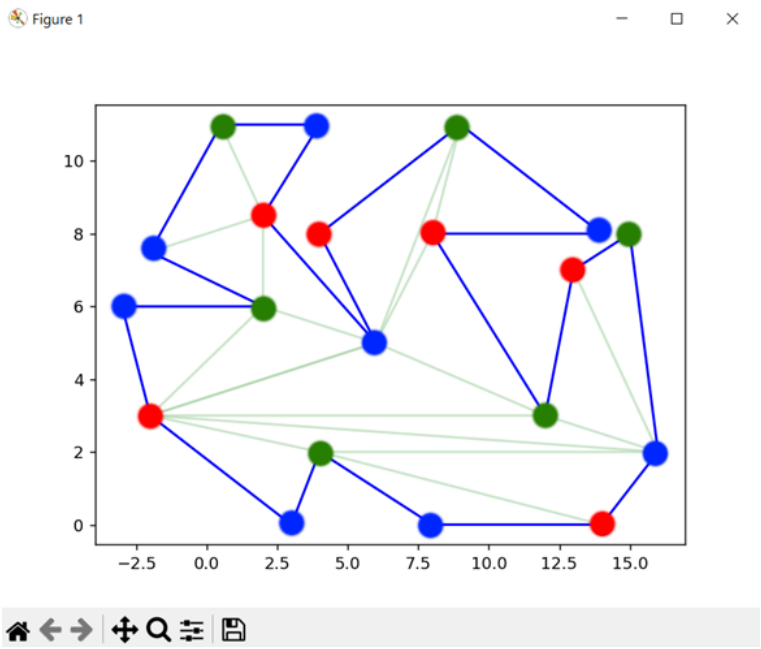

Figura 13. Coloración-3 de las triangulaciones prueba 3

- Prueba 4. Polígono de 24 lados
  Se obtuvo un resultado de 7 cámaras

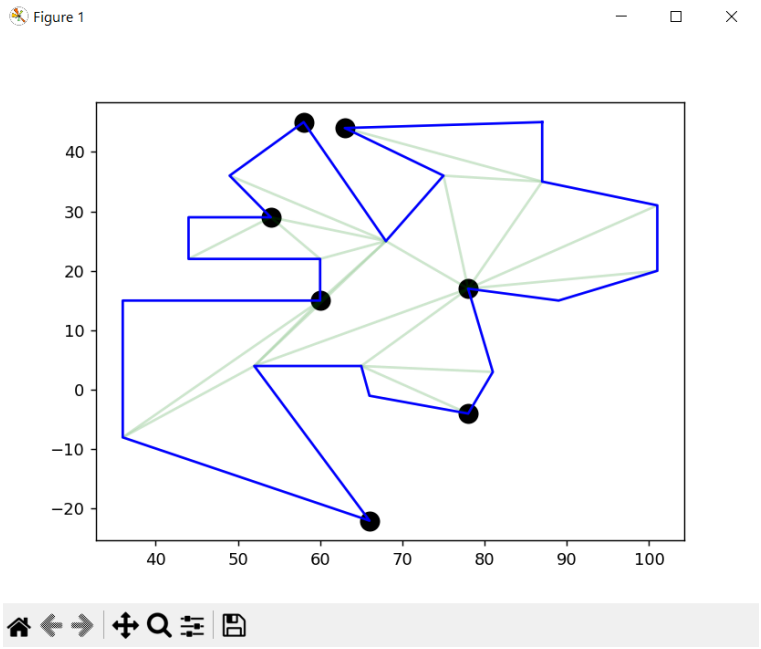

Figura 14. Resultado obtenido de la prueba 14

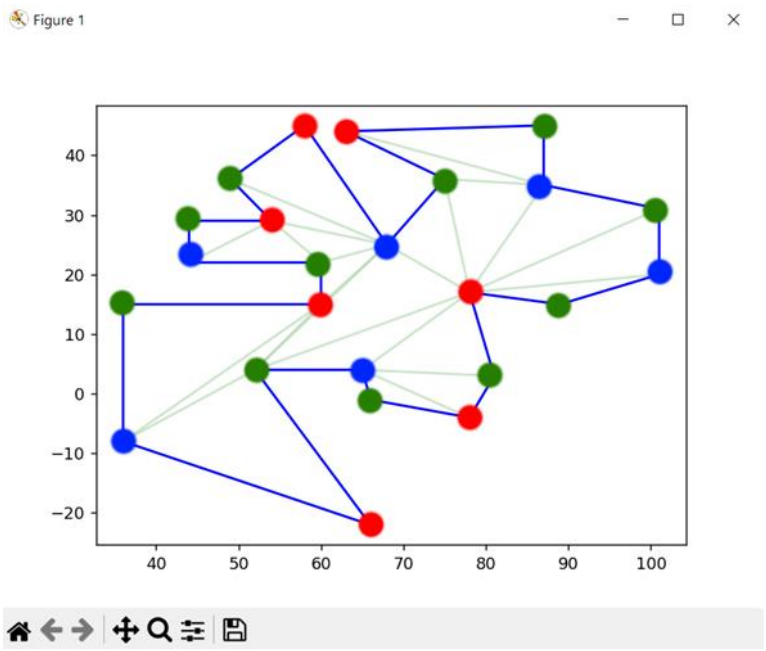

Figura 15. Coloración-3 de las triangulaciones prueba 4

Conclusión

Del presente trabajo podemos concluir que son suficientes n/3 cámaras de seguridad para cubrir un departamento, siempre y cuando el departamento pueda ser representado como un polígono simple, y las cámaras de seguridad puedan vigilar los 360 grados desde su posición.

El algoritmo utilizado para la solución del problema tiene una complejidad de $O(n^2)$. Esto se debe principalmente a dos factores:

- El algoritmo de triangulación utilizado (método de recorte de orejas) requiere de ese tiempo.
- La búsqueda del vértice más frecuente en la lista de triángulos requiere de ese tiempo.

La búsqueda del vértice mas frecuente no es necesaria para cumplir con el teorema de la galería de arte. Sin embargo, lo implemente de esta manera pensando que así se reduciría el numero de cámaras necesarias, pues al ser un vértice contenido en el mayor número de triángulos, cubriría un mayor espacio. Por falta de tiempo no logre comprobar lo anterior, así que quedara como meta futura.

Referencias

- Urrutia, J. (2004, April 28). *Art Gallery and Illumination Problems*. Matem UNAM. https://www.matem.unam.mx/~urrutia/ArtBook.html/Completo.pdf
- Bahoo, Y. (2019, September). *Computational Geometry Algorithms for Visibility Problems*. https://ponisha.ir/usercontent/400306/portfolios/143018/attachment-6a44c709023457fe612bad60ee205be1.pdf
- Wikipedia contributors. (2022, November 30). *Art gallery problem*. Wikipedia. https://en.wikipedia.org/wiki/Art_gallery_problem
- Chesnokov, N. (2018, May 16). *The Art Gallery Problem: An Overview and Extension to Chromatic Coloring and Mobile Guards*. https://math.mit.edu/~apost/courses/18.204_2018/Nicole_Chesnokov_paper.pdf
- *Art Gallery Problem | Brilliant Math & Science Wiki.* (n.d.). https://brilliant.org/wiki/guarding-a-museum/
- Eberly, D. (2022, July 5). *Triangulation by Ear Clipping.* Geometric Tools. https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf
- Wikipedia contributors. (2022a, October 23). *Polygon triangulation*. Wikipedia. https://en.wikipedia.org/wiki/Polygon_triangulation
- Wikipedia contributors. (2021, May 24). *Two ears theorem*. Wikipedia. https://en.wikipedia.org/wiki/Two_ears_theorem
- Linuxlewis. (2019, April 16). *Simple polygon triangulation algorithms in pure python*. GitHub. https://github.com/linuxlewis/tripy
- *matplotlib.pyplot — Matplotlib 3.5.3 documentation*. (n.d.). https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html