Jose Alejandro Montana
Andrea Barraza
Geometría computacional
Universidad Javeriana
Bogotá, 20 de febrero 2024

Taller 1: Intersección de polígonos

En el presente documento se plasma el análisis, diseño y consideraciones técnicas del algoritmo implementado para hallar el polígono de intersección entre dos polígonos sin huecos.

1. Análisis

Esta sección detalla el enfoque adoptado para desarrollar la lógica teórica que fundamenta el algoritmo definitivo. Inicialmente se abordó el problema mediante la exploración de diversos escenarios, identificando en cada uno de ellos posibles contradicciones o limitantes. Este proceso iterativo de refinamiento nos permitió establecer una estrategia para determinar la intersección entre dos polígonos.

Es de destacar que en las secciones 1.1 y 1.2 se enfocó específicamente en la identificación de puntos de intersección entre los polígonos. Una vez obtenidos estos puntos, el desafío consistió en determinar un método apropiado para tener los puntos organizados 1.3, con lo cual nos permitió tener un polígono final que estuviera orientado con las manecillas del reloj. Este ultimo paso fue importante para poder al final hacer el cálculo del área resultante dela intersección.

1.1. Acercamiento inicial

En un primer intento por llegar a la resolución del problema, pensando en que usar un algoritmo de fuerza bruta seria muy costoso computacionalmente y teniendo en cuenta que los polígonos tienen como partes fundamentales los vértices, el grupo de trabajo empezó considerando intersecciones de polígonos donde *al menos* uno de los vértices de un polígono estuviera contenido dentro del otro:

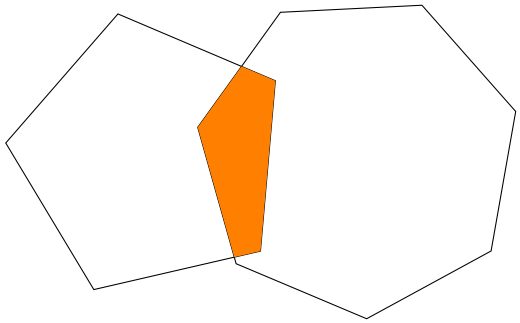

Figura 1: Intersección de dos polígonos

Supongamos que este vértice de interés fuera v_1, se pensó de manera inicial que se podría partir de dicho v_1 hacia cualquiera de los siguientes vértices del mismo polígono y empezar a preguntar a cada arista si se intersecta con *algo* (donde este algo seria el otro polígono), de esta manera nos ahorraríamos el tiempo de hacer un algoritmo de fuerza bruta donde a cada arista se le preguntara si se intersecta con *algo*. Este algoritmo fue facilmente mejorado al encontrar polígonos que se intersectan sin necesidad de que ninguno de sus vertices se intersecten: Hasta

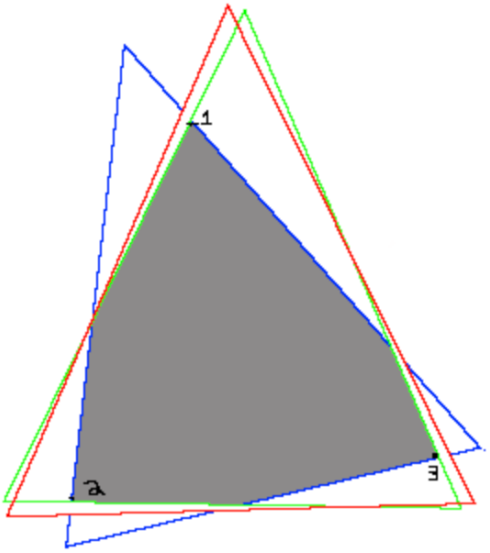

Figura 2: Intersección de tres polígonos

este momento se podría considerar que el algoritmo podría ir de la siguiente manera:

1. Preguntar si un polígono A tiene vértices de un polígono B
2. Preguntar si un polígono B tiene vértices de un polígono A

1.2. Encontrar los puntos de intersección de dos polígonos

Aunque en la sección 1.2 notamos que la idea de considerar que el vértice de un polígono A vértice este dentro de un polígono B (y vicecersa) no es suficiente para el algoritmo si pudimos notar que es necesaria. Para esto introdujimos una tercera noción que vimos que pasaba en cada caso que considerábamos (para efectos del documento se pude observar en la figura 1 y la 2), esta noción era la de empezar con un polígono (sea A o B) y escoger uno de sus lados para después preguntarte si este es intersectado por alguno de los lados del segundo polígono. En caso de que no haya intersección se puede seguir al siguiente lado del polígono pero en caso de que nos encontremos con alguna intersección deberíamos ser capaces de guardar dicho punto.

Si las intersecciones son vértices ya las tenemos guardadas por lo visto en 1.1 pero de lo contrario deberíamos tener alguna manera de guardar este nuevo punto de quiebre al que llamaremos I_p durante el resto del documento. Para este momento se pensó que se debería tener cierto sentido de orientación y se uso la misma lógica del convex hull (algoritmo incremental) para hacer un gift wrapping de estos puntos de evento donde hay intersección. Es decir, se iba a empezar en el lado l_i del polígono A y se iba a preguntar si en dicho l_i hay un I_p, en caso de que no haya se pasa al l_{i+1} en sentido de las manecillas del reloj sin hacer nada y en caso de que haya, se guarda dicho I_p y el algoritmo se dirije al vértice mas cercano a I_p que sea del polígono B y en sentido contrario al inicial (en este caso contra las manecillas del reloj) recorre ese lado s_i y vuelve a hacer la misma pregunta repitiendo lo anterior.

En pocas palabras, lo que se busca con nuestro tercer paso es empezar en cualquiera de los dos polígonos en un lado cualquiera e ir preguntando en un sentido escogido por los puntos de evento, en caso de haber un punto de evento se cambia el sentido y el polígono por el que se pregunta si hay punto de evento. Eventualmente, va a llegar un momento en el que se recorren ambos polígonos y la fase 3 debe terminar con todos los puntos de evento guardados.

1.3. De los puntos de intersección al polígono de interseccion

El algoritmo anteriormente descrito se centra en calcular la intersección entre dos polígonos. La meta es determinar el polígono resultante de esta intersección, lo cual implica identificar los puntos donde los bordes de los dos polígonos originales se cruzan, y luego organizar estos puntos de manera que formen un nuevo polígono que represente la zona común a ambos.
La ejecución del algoritmo comienza con la lectura de los dos polígonos de entrada. Se inicializa un polígono de intersección vacío y se establece un flujo de trabajo que permite iterar sobre los segmentos de cada polígono, buscando puntos donde se cruzan con el otro polígono. La búsqueda de intersecciones se realiza de manera alternada entre los dos polígonos para asegurar

una exploración exhaustiva de todos los segmentos posibles. A continuación se muestra un simple esquema del pseudocódigo que tiene esta implementación.

```
Funcion Main:
    Leer poligono_1 y poligono_2
    Inicializar poligono_interseccion, vacio
    Iteramos en sentido positivo
    Empezamos por la primera arista del poligono 1

    while encuentre intersecciones nuevas:
        para cada segmento del poligono 1
        busque intersecciones con poligono 2
        Si interseccion:
            cambiar de direccion
            cambie a poligono 2

    Calcule areas de cada poligono
    Calcule porcentaje de interseccion
```

Durante la iteración, si se encuentra un punto de intersección que no está ya en el polígono de intersección, se añade este punto. La adición de puntos se hace siguiendo una orientación específica y verificando si el punto está dentro del área de interés, para asegurar la correcta formación del polígono de intersección. El proceso continúa hasta que no se encuentran nuevos puntos de intersección, lo que indica que el polígono de intersección está completo. Cabe recalcar que este procedimiento puede fallar en el caso en que uno de los polígonos esté completamente inscrito en el otro, pues este algoritmo revisa si hay puntos de intersección para luego verificar si los segmentos en donde se produce la intersección están dentro del otro polígono. Una vez identificados todos los puntos de intersección, el algoritmo realiza una limpieza final del polígono resultante, eliminando puntos duplicados y, si es necesario, reorganizando los puntos para reflejar la apropiada orientación de los puntos en el polígono de intersección. Finalmente, se calcula el área del polígono de intersección y se compara con las áreas de los polígonos originales para obtener un porcentaje que representa la magnitud de la intersección respecto a los polígonos iniciales.

1.4. Encontrar el área del poligono de interseccion

La idea detras de esta seccion es utilizar los puntos del poligono y el teorema de stokes de la siguiente manera:

$$\int_{|\gamma(s)|} dx \wedge dy + ydx \wedge dz = \iint_{[a,b]\times[c,d]} f(s,t)\, ds\, dt \tag{1}$$

El lado izquierdo de la ecuación nos esta diciendo que la integral sobre el perímetro de la forma diferencial (que en nuestro caso serian los segmentos a recorrer) es equivalente a la integral sobre el área a considerar (en nuestro caso se puede asumir que estamos en $\mathbb{R}^2$ y cobra sentido integrar sobre un cuadrado), teniendo en cuenta esto el encontrar el área del polígono de intersección se hace equivalente a calcular:

$$\frac{1}{2}\int_{[a,b]\times[c,d]} x\partial y - y\partial x \tag{2}$$

Notese que en nuestro caso, podemos cambiar la integral por una sumatoria debido a que siempre hemos supuesto que los vértices tienen una orientación y fácilmente podríamos hacer una partición del área a integrar en pequeños triángulos (que igualmente son acotados y este es el único requerimiento para la frontera) y una vez teniendo en cuenta este cambio de integral a suma tenemos que calcular el área se convierte en:

$$\sum_{(x_0,y_0),(x_1,y_1)\in S} x_0 * y_1 - x_1 * y_0 \tag{3}$$

donde S son los segmentos del polígono y (x_i, y_i) las coordenadas de los vértices.

La ecuación 3 es la que fue implementada en el código bajo la salvedad que siempre se encapsulaba la sumatorio en un valor absoluto en caso de que la orientación del polígono fuera contra las manecillas del reloj.

1.5. Casos adicionales

Total inscripcion de un poligono dentro del otro: Se aborda debido a que la pregunta 1 o 2 de la seccion 1.1, si la cantidad de vértices es igual a la cantidad de vértices del polígono entonces esta totalmente inscrito

2. Consideraciones técnicas del algoritmo

El algoritmo de intersección de polígonos que describimos se basa en una serie de pasos computacionales y lógicos diseñados para identificar y construir un polígono que represente la zona común entre dos polígonos dados. A continuación, detallamos los aspectos técnicos clave de este proceso:

2.1. Entrada de Datos

El algoritmo comienza con la lectura de dos polígonos, denominados "polígono 1ƨ "polígono 2", desde archivos de entrada con formato .obj. Cada polígono se compone de una serie de puntos o vértices que definen su perímetro y que deben estar orientados contra las manecillas del reloj.

2.2. Inicialización

Se inicializa un objeto "polígono intersecciónçomo un tipo de dato de CGAL polygon 2 vacío. Esta es una estructura de datos diseñada para almacenar los vértices de un polígono y poder acceder a métodos específicos. Además, se establecen variables de control como dirección de giro y un iterador "segmento actual polígono"que determina en que lugar del polígono estamos revisando posibles intersecciones

2.3. Búsqueda de Intersecciones

El núcleo del algoritmo se ejecuta en un bucle que continúa hasta que no se pueden encontrar más puntos de intersección. Dentro de este bucle:
Se itera sobre los segmentos del polígono 1, buscando intersecciones con el polígono 2. Si no se añaden nuevos puntos al "polígono intersección", el algoritmo cambia de dirección e itera sobre el polígono 2, buscando intersecciones con el "polígono 1". Este proceso se repite, alternando entre los polígonos, hasta que no se encuentran nuevos puntos de intersección, momento en el cual se considera que el polígono de intersección está completo.

2.4. Manejo de Intersecciones

Para cada segmento de los polígonos, el algoritmo identifica los puntos de intersección con el otro polígono. Se determina el punto de intersección más cercano y, si este punto aún no está en "polígono intersección", se añade. La adición de puntos se realiza considerando la orientación del segmento y verificando si el punto está dentro del área de interés para asegurar la correcta formación del polígono de intersección.

2.5. Finalización y Limpieza

Una vez completada la búsqueda de intersecciones, el algoritmo realiza una limpieza de "polígono intersección.[el]iminando puntos duplicados y, si es necesario, reorganizando los puntos. Esta etapa es crucial para asegurar que el polígono resultante refleje adecuadamente la zona de intersección entre los dos polígonos originales y que la orientación sea la adecuada.

2.6. Cálculo del Área de Intersección

El algoritmo concluye con el cálculo del área del polígono de intersección. Este paso implica aplicar un método de cálculo de áreas para polígonos, basado generalmente en la fórmula de Gauss o un enfoque similar, que toma en cuenta la disposición y orientación de los vértices del polígono.