Capítulo 4

Modelado de la geometría

En el apartado anterior se han obtenido como resultado las huellas de los distintos sensores de los satélites. Para los satélites considerados en este proyecto (heliosíncronos) y regiones de interés en latitudes medias bajas (que serán también el tipo de región considerada en el proyecto) las huellas resultantes son aproximadamente rectangulares. Si las regiones de interés son también poligonales, la intersección entre las huellas y la región de interés son también polígonos; estas intersecciones se denominarán adquisiciones. Para mayor facilidad se trabajará en proyección cilíndrica equidistante, de tal manera que la longitud y latitud calculadas se corresponden directamente con las coordenadas x e y de las gráficas.

Para la resolución del problema es necesario realizar intersecciones, uniones y diferencias entre polígonos. Para ello se ha utilizado la herramienta de geometría computacional PolygonClip, ejecutable desde Matlab. El comando toma como entradas dos estructuras (o vectores de estructuras) de Matlab y un número entero de 0 a 3. Las estructuras de Matlab contienen los datos sobre los dos polígonos sobre los que se va a realizar una operación, que se selecciona mediante el entero. Las estructuras deben tener tres campos:

- *.x*: vector conteniendo las abscisas de los vértices del polígono.
- *.y*: vector conteniendo las ordenadas de los vértices del polígono.
- *.hole*: entero, puede tomar el valor 0 o 1.

El funcionamiento del campo *.hole* es un tanto complejo. Primero es necesario aclarar que en una misma variable puede haber más de un polígono, y en ese caso se almacena como un vector de estructuras. Por ejemplo, los dos triángulos de la figura podrían estar almacenados en la variable $triangle$, un vector de dos componentes donde cada una

de ellas es una estructura con los campos mencionados antes. La primera componente correspondería al triángulo azul y la segunda al rojo.

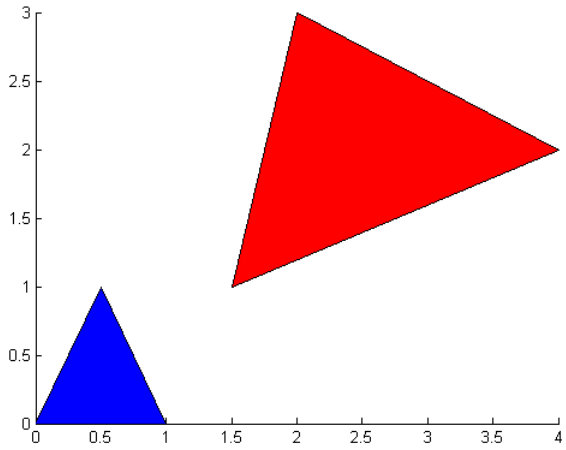

Figura 4.1: Triángulos almacenados en una misma estructura

El campo *.hole* toma el valor 0 si el polígono definido por sus coordenadas es conexo y 1 si es un agujero. No es realmente conexo, ya que un polígono como el de la figura estaría almacenado como un vector de dos componentes. La primera componente correspondería al polígono exterior y tendría un 0 en *.hole*, y la segunda componente tendría los vértices del agujero interior y un 1 en *.hole*. Cabe resaltar que el orden de las componentes no tendría por qué ser ese, es decir, el agujero podría estar almacenado en la primera componente también.

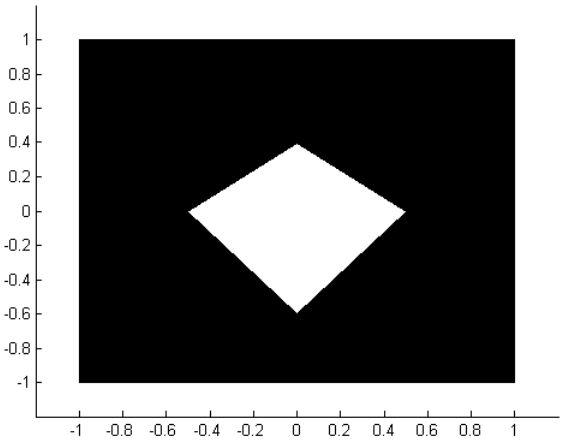

Figura 4.2: Polígono con agujero

Aunque pueda parecer obvio, el orden de los vértices en los campos es importante. PolygonClip no distingue en cuanto a sentido horario o antihorario de los vértices, pero como se puede observar en la figura, es importante recorrerlo en uno de esos dos sentidos y no salteados.

Las operaciones que se pueden realizar con PolygonClip son:

- 0: diferencia, resta el segundo polígono del primero.
- 1: intersección entre los dos polígonos.
- 2: XOR, el polígono resultante está formado por las partes de los dos polígonos que no intersectan entre ellos.
- 3: unión de los dos polígonos.

Si el resultado de una de las operaciones es un conjunto vacío, PolygonClip devuelve un vector de dimensiones 1x0.

4.1. Simplificación de las regiones

Debido a que uno de los requisitos fundamentales es el tiempo de ejecución del algoritmo, es conveniente ahorrar tiempo allí donde se pueda. Para reducir el número de puntos a manejar por PolygonClip a la hora de realizar operaciones con las adquisiciones, las regiones se someten a un proceso de simplificación.

En primer lugar se simplifican las huellas de los sensores. Como se explicó antes, al tratarse en su mayoría de satélites heliosincronos y de regiones en latitudes no extremas (alejadas de los polos), las trazas son prácticamente asemejables a rectángulos. Al propagarse las órbitas con un intervalo de tiempo fijo, se obtienen unas regiones con mucho puntos superfluos al estar éstos alineados. Para ello un primer proceso de depuración consiste en verificar para cada tres puntos consecutivos P,Q y R el ángulo formado por los vectores $\overline{PQ}$ y $\overline{PR}$, de tal manera que si el ángulo es menor de una cierta tolerancia se elimina el punto Q del conjunto de puntos que definen la huella.

Más adelante, al realizar operaciones de intersección y diferencia se generan polígonos con pequeños errores, que si se dejan pueden llevar al fallo del algoritmo. Un ejemplo del error es el siguiente. Se tienen las dos adquisiciones de la figura:

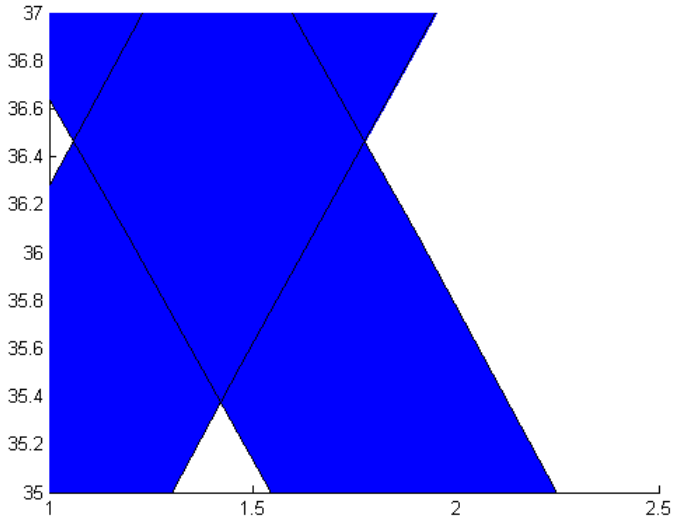

Figura 4.3: Adquisiciones que llevan a error

Cuya intersección es la de la figura siguiente:

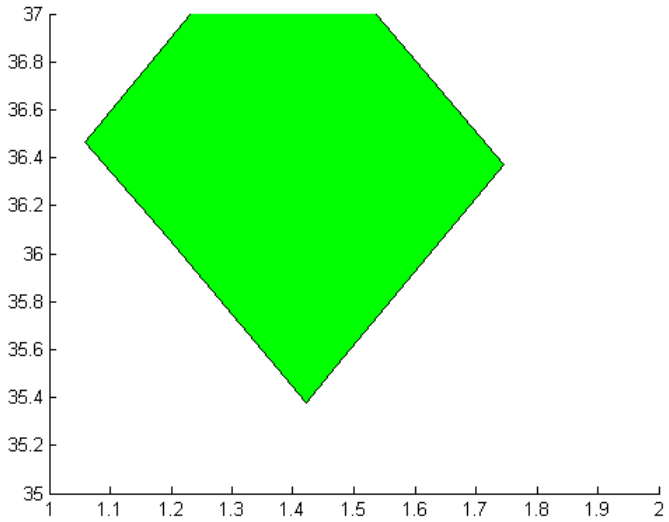

Figura 4.4: Intersección entre dos adquisiciones

Y al restar de la intersección que va de suroeste a noreste la intersección deberían quedar dos subregiones completamente separadas de tres y cinco vértices. No obstante el resultado es un único polígono que tiene la siguiente forma:

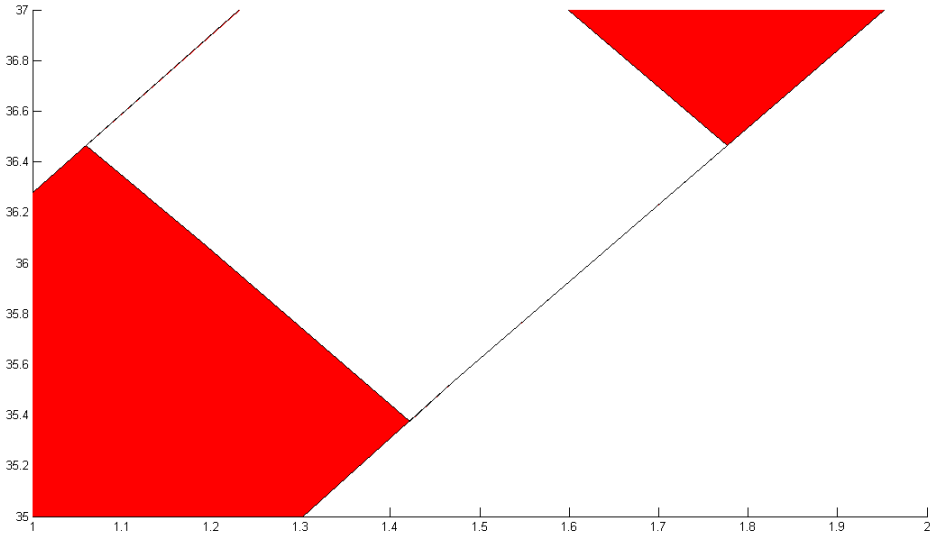

Figura 4.5: Polígono generado con el error

Se observa que los dos polígonos que deberían estar separados quedan unidos por una línea que queda a la derecha, y además a la izquierda aparece un pico que no debería aparecer. Cuando más adelante se alimentan este tipo de polígonos a PolygonClip, éste da un fallo y se cierra, cortando toda la ejecución. Este problema se detectó durante el desarrollo de [1], pero no se hizo nada por solventarlo. En este proyecto se ha implementado una rutina que comprueba las apariciones de este tipo de polígonos y elimina las irregularidades, aunque esto conlleva un tiempo mayor de ejecución. No obstante, dado el elevado número de adquisiciones con que se trabaja en este proyecto se hace necesario, ya que hasta en los ejemplos más simples aparece este error.

La rutina que se encarga de subsanar este error lo hace recorriendo cada uno de los puntos del polígono generado por la herramienta de geometría computacional y verifica para cada tres puntos consecutivos P,Q y R que el ángulo formado por los vectores $\overline{QP}$ y $\overline{QR}$ no sea menor de una tolerancia. En caso contrario significaría que existe un pico y por tanto se elimina el punto Q. Además, si resulta que los puntos P y R están muy cercanos uno al otro se elimina también uno de ellos.

Este proceso de depuración es uno de los principales culpables del tiempo de computación. Necesita recorrer todos los puntos de cada una de las regiones generadas y realizar las comprobaciones, con lo que, junto al resto de operaciones que conlleva el cálculo de la matriz Q_{ij}, consume la mayor parte del tiempo.