Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Triangularización de espacios de datos a través de la librería Hitmap

---

Curso académico:
2020-2021

Alumno:
Hugo Prieto Tárrega

Tutores académicos:
Yuri Torres de la Sierra
María Inmaculada Santamaría Valenzuela

A Imaz.

Agradecimientos

Quiero agradecer a mis padres el apoyo que me han dado durante todo mi periodo como estudiante en la universidad, sobretodo durante este último curso académico, pues sin ellos no podría haberme centrado tanto en mi desarrollo académico.

Por otro lado quiero agradecer a Yuri Torres de la Sierra y a Arturo González Escribano por sus esfuerzos por guiarme y enseñarme durante el desarrollo de este proyecto y en especial a María Inmaculada Santamaría Valenzela, por su paciencia y su buen hacer.

Resumen

Actualmente, el paradigma de la computación paralela es ampliamente utilizado por científicos e investigadores para la resolución de problemas de cálculo con gran carga computacional. En la lista de TOP 500 se refleja el ranking con los 500 supercomputadores más potentes del mundo que existen en la actualidad. Estos sistemas son empleados para realizar cálculos con grandes volúmenes de datos en diferentes áreas de conocimiento tales como las matemáticas, la astronomía, la biología o la medicina.

Dentro del paradigma de la computación paralela destacamos la técnica denominada como "Tiling". Esta técnica consiste en el reparto de la carga computacional entre diferentes unidades computaciones. Este reparto de carga se consigue a través del particionado del espacio de memoria donde se alojan los datos. Se realiza una división del dominio de datos del problema en bloques asignados a cada procesador.

Hitmap es una librería diseñada por el grupo de investigación Trasgo que aúna mecanismos de comunicación así como mecanismos del particionado de datos. Este particionado se realiza con bloques de forma rectangular modelados a través de la estructura de datos *Shape*, que permite la creación de polígonos rectangulares de n-dimensiones.

En este trabajo se desarrolla una extensión de la librería Hitmap para la triangularización del espacio de datos. Esta técnica se implementa a través de particiones triangulares de memoria permitiendo al sistema de computo paralelo aprovechar mejor la geometría del espacio de datos en ciertos tipos de problema, para reducir, en la medida de lo posible, las operaciones de comunicación y sincronización. Para realizar este particionado en polígonos triangulares se desarrollan los algoritmos de intersección, diferencia y unión.

Abstract

Nowadays, the paradigm of parallel computing is widely used by scientists and researchers for solving computational problems with high computational load. The TOP 500 list reflects the ranking of the 500 most powerful supercomputers in the world that currently exist. These systems are used to perform calculations with large volumes of data in different areas of knowledge such as mathematics, astronomy, biology or medicine.

Within the paradigm of parallel computing we highlight the technique known as "Tiling". This technique consists of the distribution of the computational load between different computational units. This load sharing is achieved through the partitioning of the memory space where the data is stored. The data domain of the problem is divided into blocks assigned to each processor.

Hitmap is a library designed by the Trasgo research group that combines communication mechanisms as well as data partitioning mechanisms. This partitioning is done with rectangular shaped blocks modeled through the *Shape* data structure, which allows the creation of n-dimensional rectangular polygons.

In this project an extension of the Hitmap library is developed for the triangularization of the data space. This technique is implemented through triangular memory partitions, allowing the parallel computing system to take better advantage of the geometry of the data space in certain types of problems, to reduce, as far as possible, the communication and synchronization operations. To carry out this partitioning in triangular polygons, the intersection, difference and union algorithms are developed.

Índice general

Índice de figuras

Índice de cuadros

Listings

Capítulo 1

Introducción

En este capítulo se introducen los siguientes aspectos:

- El contexto y la motivación del proyecto.
- El planteamiento del problema y los objetivos del proyecto.
- La organización de los contenidos del documento.

1.1. Contexto

En esta sección se describe el contexto en el que se desarrolla este proyecto, introduciendo los principales fundamentos de la computación paralela, de los sistemas paralelos, de los sistemas heterogéneos y del particionado del espacio de datos.

1.1.1. Computación paralela

Actualmente, existe una gran cantidad de problemas de carácter científico cuya resolución es muy costosa en tiempo de ejecución con los métodos de computación clásicos. Entre los ejemplos de estos problemas encontramos simulaciones que involucran cálculos complejos y/o grandes volúmenes de datos en diferentes áreas de conocimiento [1] tales como las matemáticas, astronomía, biología o medicina, que sin el paradigma de la computación paralela tardarían una cantidad significativa de tiempo de procesamiento continuado para finalizar.

Existen otras aplicaciones como el deep learning [2], basadas en el uso de redes neuronales, que sin la mejora de rendimiento que supone el uso de sistemas de computo paralelos sería prácticamente inabordables. Estas redes neuronales aprovechan todos los recursos hardware existentes en dispositivos aceleradores, como GPUs (unidades de Procesamiento Gráfico [3])

y FGPAs (matriz de puertas lógicas programable [4]). La minería de monedas criptográficas, como los BitCoins, es otro ejemplo basada en la resolución de problemas matemáticos, soportada en gran medida por las tarjetas gráficas.

Para conseguir estas optimizaciones, la computación paralela se basa en el uso de muchas unidades computacionales que llevan a cabo cálculos de forma simultánea [5]. De esta forma, grandes problemas computacionales son divididos en otros problemas más pequeños, resueltos individualmente por cada una de estas unidades de procesamiento. Un ejemplo típico de estos problemas es una multiplicación de matrices con un gran número de elementos. Los espacios de datos son repartidos entre los nodos de computo, reduciendo de forma teórica los tiempos proporcionalmente al número de nodos.

Actualmente, los ordenadores con mayor capacidad computacional del mundo emplean el paradigma de la computación paralela y se invierten millones de euros en el desarrollo y mejora de estas supercomputadoras, así como en el alquiler de las mismas para la realización de cálculos científicos. En la lista de TOP 500 [6] se publica cada medio año la lista actualizada de los 500 superordenadores más potentes del mundo. Además, también se publica la lista de los Green500, con los 500 supercomputadores energéticamente más eficientes.

1.1.2. Sistemas paralelos

Una aplicación secuencial puede escalar su rendimiento de manera proporcional al número de unidades computacionales del sistema. En este caso ideal, con dos o cuatro unidades funcionales el tiempo se ve dividido a la mitad o la cuarta parte, respectivamente. El tiempo necesario para ejecutar una aplicación de forma secuencial puede optimizarse mejorando su rendimiento de manera lineal, es decir, si se duplican sus elementos de cómputo el tiempo se reduce a la mitad del tiempo obtenido secuencialmente, si se triplican este tiempo se reduce a una tercera parte, y así sucesivamente.

Esa mejora del rendimiento es ideal, ya que en la práctica no se alcanzan estos escenarios, debido a problemas como el tiempo de encaminamiento, en concreto con los mecanismos de de comunicación y sincronización, asociados al coste temporal que supone la división del problema entre subtareas concurrentes y el paso de datos entre procesadores debido a dependencias; la necesidad de disipar el calor generado por estos grandes supercomputadores o secciones de código que solo pueden ser ejecutadas de forma estrictamente secuencial.

A pesar de los problemas anteriormente mencionados, la computación paralela presenta una serie de ventajas [7] tal y como se observan en los siguientes items:

- La resolución de problemas que no se podrían ser ejecutados secuencialmente con una sola CPU en un tiempo razonable.
- Ejecución de una mayor carga computacional en menos tiempo a través de aceleradoras hardware como GPUs y FGPAs.
- Posibilidad de ejecución de problemas de complejidad y orden mayor en un tiempo razonable.

- Permite la ejecución simultánea de varias instrucciones y/o tareas.
- Se obtiene un mejor balance entre productividad y coste que en la computación secuencial tradicional.
- Se obtiene un mejor balance entre productividad y coste energético que en la computación secuencial tradicional.
- Gran capacidad expansión, es decir, posibilidad de añadir nuevos nodos y dispositivos hardware.
- Gran capacidad de escalabilidad, es decir, que la mejora del sistema aumente de forma proporcional a la expansión en nodos.

Actualmente, el supercomputador más potente del mundo está localizado en Japón. Se trata de Fugaku [8], que cuenta con un total de 7.600.848 cores y una potencia de cálculo de 415.599 Teraflops, lo que equivale a 230.833 PlayStation 4 trabajando simultáneamente [9].

1.1.3. Sistemas heterogéneos

Un sistema de computación heterogéneo es aquel que está compuesto por unidades computacionales y/o nodos con arquitecturas distintas, conectados a través de un sistema de red. MareNostrum_5 [10] es un ejemplo de sistema heterogéneo, siendo éste uno de los supercomputadores más avanzados de Europa, localizado en España, cuyas optimizaciones más recientes consisten en la integración de una mayor variedad de procesadores, cada unos orientados a aplicaciones más específicas.

Los sistemas heterogéneos presentan una serie de ventajas [11] frente a los sistemas homogéneos, entre las que se encuentran:

- La diversidad de procesadores permite diferentes opciones de ejecución, empleando aquellos recursos mejor optimizados para las distintas tareas que conforman un problema.
- El gasto energético asociado a los cálculos puede ser regulado eligiendo qué nodos van a llevar a cabos los cálculos.
- Ante la posibilidad de presentar diferentes potencias de cálculo en cada nodo, se puede llevar a cabo repartos de carga adaptados a las velocidades de los distintos procesadores, para alcanzar un nivel superior de optimización.

1.1.4. Particionado de datos

Una de las características principales de la computación paralela es la posibilidad de repartir de carga entre los distintos procesadores o nodos de computo, de forma que las distintas tareas y conjuntos de datos son asignados a cada unidad de cómputo teniendo en

cuenta su rendimiento. Aquellos nodos con mayor productividad realizan cálculos con un volumen mayor de datos. Así se logra igualar los tiempos de ejecución entre los procesares.

Este reparto de carga se logra mediante el particionado de datos, en el cual el dominio de datos del problema, generalmente arrays multidimensionales, es divido en bloques que son asignados a los nodos de computo. Con el fin de optimizar el rendimiento de las aplicaciones paralelas, las particiones serán equitativas para aquellos sistemas cuyos procesadores tengan productividades iguales, mientras que para sistemas heterogéneos, este particionado se hará en bloques de distintos tamaños. El particionado, en última instancia busca reducir las comunicaciones y sincronizaciones y su optimización depende de la geometría del problema computado y de la alocación física de los datos.

Existen diferentes técnicas de particionado (en bloques rectangulares, en bloques que coinciden con las filas de un array bidimensional, en bloques que almacenan los datos que presentan dependencias entre sí, entre otros). Sin embargo estas técnicas no consiguen reducir la cantidad de datos a transferir. El particionado triangular reduce esta cantidad para ciertos tipos de problema de cómputo aprovechándose de la geometría subyacente del espacio de datos.

1.2. Motivación

1.2.1. Grupo de investigación Trasgo

Este Trabajo de Fin de Grado se desarrolla en el contexto de uno de los proyectos del grupo de investigación Trasgo [12]. Este grupo pertenece al Departamento de Informática de la Universidad de Valladolid [13].

A continuación se muestran las áreas de investigación principales en las que trabaja dicho grupo de investigación:

- El desarrollo de modelos de programación, sistemas de ejecución y herramientas para sistemas heterogéneos paralelos.
- La paralelización especulativa.
- Búsqueda de soluciones paralelas a problemas computacionalmente intensivos.
- El desarrollo de software de sistema para aplicaciones embebidas, utilizando tecnología de código abierto.

Actualmente, se está desarrollando la librería Hitmap [14], que permite el particionado del espacio de datos en bloques rectangulares. Existen modelos de particionamiento relativamente sencillos basados en la división de la memoria de datos en bloques rectangulares de dimensiones bxh, y el siguiente paso es la triangularización, mediante la cual se podrá aprovechar mejor la geometría del problema.

1.2.2. Trabajo relacionado

Tiling [14] es una técnica empleada para distribuir datos y tareas en programas paralelos y mejorar la localidad (almacenamiento de datos relacionados en posiciones contiguas de memoria [15]) de bucles anidados en código paralelo y secuencial. El uso de estructuras de datos para implementar particiones en matrices de memoria genéricas permite explotar mejor la jerarquía de memoria, haciendo que los datos sean a menudo reutilizados dentro de la misma partición.

El particionado de datos se puede aplicar de forma anidada (la matriz de memoria se divide en bloques que a su vez pueden ser divididos en subloques y así sucesivamente), teniendo por objetivo distribuir el trabajo entre los procesadores en el nivel más externo o mejorar la localidad en el nivel más interno. Al trabajar con memoria distribuida, las particiones también pueden hacer que la comunicación sea explícita, ya que los cálculos que involucran elementos de diferentes particiones dan como resultado el movimiento de datos, algo indeseable en términos de rendimiento.

Durante la última década, se han propuesto diferentes modelos de programación para manejar la complejidad de la partición y mapeo de datos multinivel. Estos modelos de programación se dividen en dos categorías: aquellos que ocultan las comunicaciones subyacentes y aquellos donde la comunicación explícita es impulsada por la partición hecha por el usuario (por ejemplo, MPI). Estos modelos de programación paralela no ayudan al programador a expresar explícitamente el patrón de comunicación que necesita el algoritmo, independientemente de la partición de datos elegida. Bajo este pretexto, el grupo Trasgo ha desarrollado la librería Hitmap.

1.2.3. Librería Hitmap

Hitmap [14] es una librería diseñada por el grupo Trasgo para desacoplar el patrón de comunicación del particionado de datos, mediante el uso de expresiones abstractas que se adaptan automáticamente en tiempo de ejecución dependiendo de la partición finalmente utilizada.

Hitmap presenta una combinación de características tales como:

1. Un sistema de plug-in's (complementos que agregan nuevas funciones a un programa anfitrión sin alterar al mismo [16]) extensibles, basados en dos tipos de módulos diferentes, uno para calcular automáticamente el particionado de datos y otro para la distribución de particiones en función de la topología de la arquitectura subyacente, ocultando los detalles al programador.
2. Un framework (plataforma para el desarrollo software [17]) para programar nuevos plug-in's con técnicas regulares, irregulares, estáticas o dinámicas de partición y equilibrio de carga.
3. Un conjunto de herramientas flexible para el mapeo de tareas paralelas y datos con una interfaz común.

4. Una API para crear patrones de comunicación complejos y escalables en términos de una partición y un diseño abstractos.

Todas estas características permiten integrar decisiones de mapeo complejas, algunas de ellas asociadas a la tecnología de compilación, en una biblioteca.

Hitmap se puede utilizar para admitir estructuras de datos complejas, como matrices dispersas y gráficos para aplicaciones irregulares, utilizando la misma metodología de particionado jerárquico, o para programar un sistema heterogéneo [18]. El resultado es un buen equilibrio entre rendimiento y uso eficiente de la memoria, lo que también reduce el esfuerzo de programación en comparación con otras opciones.

Una de las carencias que presenta Hitmap es la imposibilidad para adaptarse a problemas de cómputo cuya geometría es algo más compleja y el particionado del espacio de datos mediante bloques rectangulares o cúbicos no permite a los sistemas de cómputo aprovechar al máximo las mejoras obtenibles mediante el paralelismo.

1.2.4. Triangularización de espacios de datos

A la hora de particionar el espacio de datos de un problema surge la necesidad de establecer comunicaciones y sincronizaciones entre los diferentes datos cuando que tengan dependencias entre si la información manejada por los distintos nodos de procesamiento.

Para el ejemplo de una simulación de incendios (práctica de la asignatura de Computación Paralela del grado de Ingenieria Informática en la UVa del año 2018-2019 [19]), donde la forma en la que se extiende el fuego depende de áreas contiguas, en la siguiente imagen se puede apreciar como la partición cuatro necesita sincronizarse y comunicarse con las particiones que manejan los datos de estas respectivas zonas.

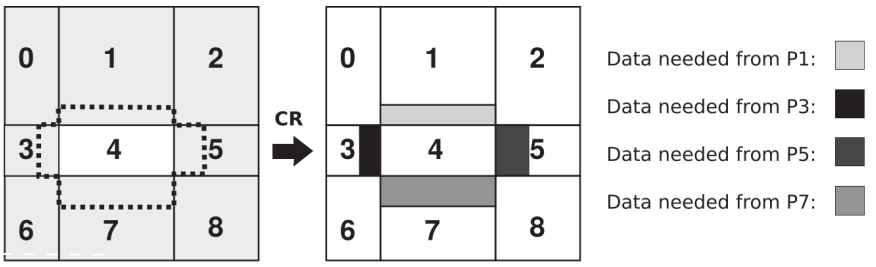

Figura 1.1: Particionado de datos con particiones rectangulares (Figura extraída de [20]).

Los mecanismos de sincronización y comunicación de las particiones asignadas a los distintos nodos de computo suponen uno de los principales cuellos de botella de la computación paralela. La mejora de rendimiento que supone este paradigma frente a la computación secuencial tradicional es afectada sustancialmente por el número de veces que se emplea alguno

de estos mecanismos a lo largo de la ejecución de un programa paralelo y es por esto que es fundamental reducir las llamadas a estas rutinas.

La triangularización del espacio de datos es una técnica que permite reducir las comunicaciones y sincronizaciones, adaptándose mejor a la geometría del problema que resuelve el sistema de cómputo. Este mejor uso de la geometría del espacio de datos se logra a través del particionado, ya explicado en la sección 1.1.4. La principal diferencia del particionado de memoria clásico es la admisión de triángulos como unidad de división frente al uso de particiones rectangulares para dos dimensiones o cúbicas para tres dimensiones.

La posibilidad de emplear triángulos para el particionado, en adición con el particionado clásico, permite evitar que un mismo dato sea gestionado redundantemente por más de un procesador, así como favorecer que todos los índices de una partición gestionen datos en vez de quedar *ociosos*, aumentando la productividad en términos de información calculada por bloque (ya sea rectángular, triangular, piramidal o cúbico).

La resolución de múltiples sistemas de ecuaciones por el método de Gauss de forma concurrente al obtener el sistema equivalente de un sistema de ecuaciones mediante la eliminación gaussiana simple es un ejemplo de problema computacional que se ve favorecido por el uso de la triangularización. Pongamos como ejemplo el siguiente sistema de ecuaciones:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 1 \\ 4x_1 + 5x_2 + 6x_3 &= -2 \\ 7x_1 + 8x_2 + 10x_3 &= 5 \end{aligned}$$

Figura 1.2: Sistema de 3 ecuaciones con tres incógnitas.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 1 \\ 3x_2 + 6x_3 &= 6 \\ x_3 &= 10 \end{aligned}$$

Figura 1.3: Sitema de ecuaciones equivalente por elminacipón Gaussiana.

En la figura 1.4 se muestra que al utilizar el bloque clásico rectangular (rectángulo verde) como unidad de particionado el bloque clásico rectangular (rectángulo verde) existen tres índices de memoria (triángulo rojo) de un total de 12, manejados por la unidad de computo responsable de la partición, cuyos datos y cálculos no son necesarios para la resolución del problema y por tanto se limita la mejora de rendimiento de la paralelización.

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & 3 & 6 & 6 \\ 0 & 0 & 1 & 10 \end{array}\right)$$

Figura 1.4: Sistema de Gauss resuelto con particiones rectangulares.

En la figura 1.5 se muestra como la triangularización del espacio de datos se adapta

mejor que el particionado clásico a la geometría de este problema, de forma que solo una posición de memoria de las diez gestionadas en la partición triangular no es necesaria para la resolución del problema y por tanto, la mejora del rendimiento obtenida por el paradigma de la computación paralela es mayor que el obtenido hasta el momento. Esto se debe a que el número de particiones que podrán realizarse será mayor, harán un mejor uso del espacio de datos y por ende, se reducirán las comunicaciones y sincronizaciones.

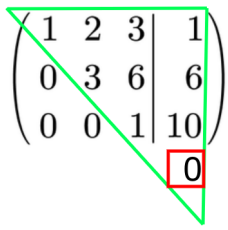

Figura 1.5: Sistema de Gauss resuelto con particiones triangulares.

1.3. Objetivos

1.3.1. Objetivo del proyecto

En este proyecto se propone el desarrollo de un mecanismo de particionado del espacio de datos de memoria en bloques triangulares empleando n-símplices (estructuras triangulares) con lados paralelos a los ejes de coordenadas [21]. Este particionado se incluirá como extensión de la librería Hitmap, desarrollada por del grupo de investigación Trasgo, partiendo de la estructura de datos HitSigShape, que será analizada más adelante en la sección 3.1. Para operar con las nuevas estructuras se analizará, diseñará e implementará las operaciones binarias de intersección, unión y resta y sus algoritmos respectivos.

Este Trabajo de Fin de Grado se desarrolla dentro del contexto de un proyecto de investigación, con su correspondiente ciclo de desarrollo, y es por ello que tiene asociado una serie de objetivos adicionales. Además, el código implementando será usado por el grupo de investigación Trasgo una vez haya sido validado e integrado con sus librerías, por ende también surge una serie de características deseables para el software desarrollado:

- Debe ser comprensible y estar correctamente documentado.
- Ser extensible y reutilizable.
- Debe estar bien estructurado en subfunciones, con el fin de evitar duplicidad de código, además de facilitar el seguimiento del flujo del programa.
- Debe utilizar la notación y los paradigmas de programación del grupo Trasgo.

- Debe estar bien preparado para cambios futuros.
- Debe ser correcto en sus cálculos.

1.3.2. Objetivos secundarios

Este trabajo se divide en una serie de objetivos parciales para cumplir con el objetivo principal del proyecto:

- Estudiar, analizar y comprender las principales estructuras de Hitmap.
- Estudiar, analizar y comprender las operaciones binarias de intersección, unión y diferencia entre dos polígonos para un espacio de dos dimensiones.
- Diseñar e implementar las estructuras de datos extendidas de la librería Hitmap para poder modelar particiones con formas triangulares.
- Diseñar e implementar la operación de intersección entre dos rectángulos, dos triángulos o un triángulo y un rectángulo.
- Diseñar e implementar la operación de unión entre dos rectángulos, dos triángulos o un triángulo y un rectángulo.
- Diseñar e implementar la operación de diferencia entre dos rectángulos. dos triángulos o un triángulo y un rectángulo.
- Extender las operaciones de intersección, diferencia y unión para que se ejecuten para un número n de polígonos (triángulos o rectángulos).
- Diseñar, elaborar y ejecutar un plan de pruebas que valide la corrección de los cálculos realizados por los algoritmos.
- Analizar los resultados y solventar aquellos errores surgidos de las pruebas de validación.
- Elaborar un documento que refleje de forma clara y precisa la naturaleza del proyecto, su estructura, el proceso seguido y las conclusiones extraídas.

1.4. Estructura del documento

Este documento está estructurado en los siguientes capítulos:

- El capítulo dos explica la planificación del proyecto.
- El capítulo tres describe las estructuras utilizadas por la librería Hitmap y el diseño de los algoritmos de intersección, unión y diferencia.

- El capítulo cuatro explica la implementación final de los algoritmos mencionados, a través de sus cabeceras, includes y funciones principales.
- El capítulo cinco presenta las pruebas realizadas para validar los resultados obtenidos por los tres algoritmos.
- El capítulo seis describe las conclusiones, incluyendo objetivos cumplidos y trabajo futuro.

Capítulo 2

Planificación

En este capítulo se introducen los siguientes aspectos:

- El modelo de desarrollo empleado para llevar a cabo el proyecto.
- Los riesgos del proyecto, junto al camino crítico y el plan de contingencia para los distintos riesgos.
- El coste estimado del desarrollo del proyecto.

2.1. Modelo de desarrollo

Para el desarrollo de este proyecto se ha escogido como método un System Development Life Cycle [22] (a partir de ahora referenciado como SDLC) lineal en cascada. En este SDLC el proyecto es divido en una serie de fases secuenciales, en las que se enfatiza el diseño y cumplimiento de un plan, con unas fechas objetivo y donde se realiza un control riguroso durante el ciclo de desarrollo sobre el avance y estado del proyecto mediante documentación y reuniones periódicas con los "project managers". En este proyecto, los encargados de la supervisión del desarrollo son los tutores académicos, Yuri Torres de la Sierra y María Inmaculada Santamaría Valenzuela.

Este modelo de desarrollo ha sido seleccionado por ser un buen candidato para proyectos donde el equipo de desarrollo no cuenta con mucha experiencia, debido a la rigurosidad con la que se comprueba el alcance de los objetivos establecidos, así como la corrección de la documentación y el software desarrollado, además de que permite medir con facilidad el avance del desarrollo. Además, es fácilmente integrable con las mecánicas de trabajo del grupo de investigación de Trasgo, con reuniones de trabajo semanales y diversos canales de comunicación directa, que permiten realizar reportes con facilidad.

2.1.1. Fases del desarrollo

Inicialmente, el desarrollo del proyecto se planifica para llevarse a cabo en ocho fases principales. Estas ocho etapas son secuenciales, como consecuencia de la metodología de trabajo seleccionada, y la suma de las duraciones de cada una de ellas da como resultado una duración estimada de 300 horas, correspondientes con los 12 créditos ECTS asociados a la asignatura del TFG.

Las fases en las que se ha dividido el proyecto vienen reflejadas en la siguiente tabla:

ID	Descripción
F1	Estudio de la librería Hitmap.
F2	Estudio de los algoritmos.
F3	Toma de decisiones de diseño relacionadas con el trabajo previo del Technical Report [21].
F4	Implementación del algoritmo de intersección.
F4.1	Implementación del algoritmo de construcción del polígono de intersección.
F4.2	Implementación del algoritmo de partición del polígono de intersección.
F5	Implementación del algoritmo de la diferencia.
F6	Implementación del algoritmo de la unión.
F7	Experimentación.
F8	Elaboración de la memoria.

Cuadro 2.1: Fases del desarrollo de proyecto.

Las fases F1, F2 y F3 están relacionadas con el estado del arte del proyecto, aprendizaje de las tecnologías y lectura del trabajo relacionado previo. Estas son proyectadas para la segunda mitad del primer cuatrimestre, invirtiendo una cantidad de horas semanales menor que el dedicado durante el segundo cuatrimestre debido a la necesidad de compaginar el trabajo con otras cinco asignaturas. Por su parte, la tareas F4, F5, F6, F7 y F8 se proyectan para el segundo cuatrimestre, con un carga semanal aproximada de trabajo de 20 horas.

Las fases F4, F5 y F6 se corresponden con la implementación de la extensión de la librería Hitmap, desarrollando los algoritmos de intersección, unión y diferencia. La fase F4, es la de mayor duración del proyecto porque engloba aproximadamente un 80 % de la programación total del trabajo, puesto que, como se mencionará en el capítulo 3, los algoritmos de unión y diferencia están basados en el de intersección. Dentro de la etapa F4 existen dos subfases, correspondientes a la descomposición principal del flujo del algoritmo de intersección, la construcción del polígono de intersección y la partición del polígono de intersección.

Tras las fases del estado del arte, desarrollo e implementación, se proyectan las dos últimas etapas del proyecto, relacionadas con la experimentación y la documentación. La fase F7 se corresponde con la realización de la experimentación para validar la corrección de los

algoritmos, y la F8 con la elaboración de este documento. Para estas tareas se estima el último de mes de trabajo.

A continuación, se muestra el diagrama de Gantt [23], que modela el reparto temporal de cada una de las etapas:

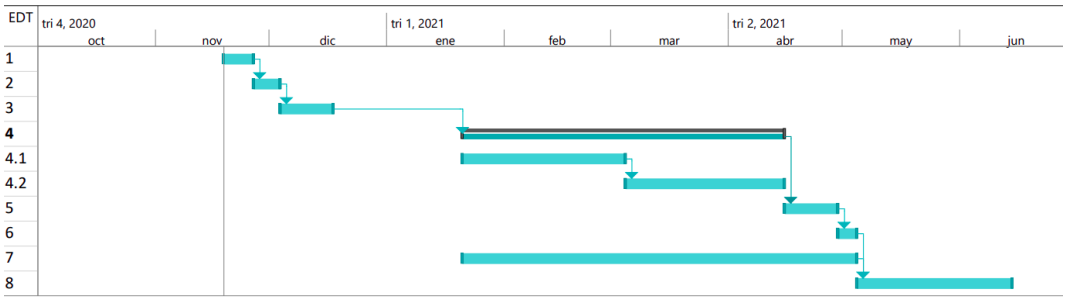

Figura 2.1: Diagrama de Gantt del proyecto.

El número que figura en la columna de la izquierda en el diagrama de Gantt 2.1 se corresponde con el ID de la etapa de desarrollo. Para la primera etapa, iniciada en la segunda mitad de noviembre se planifica para ser llevada a cabo en una semana. Tras esta, siguen la etapa 2 y 3, realizadas de forma secuencial, con una duración una y dos semanas, respectivamente. Tras estas tres primeras fases, relacionadas con el estado del arte del proyecto, se produce una parada en el desarrollo hasta mediados de enero, con la finalización del período de exámenes del primer cuatrimestre.

Con la finalización del primer cuatrimestre del curso académico 2020/21, se reanuda el desarrollo del proyecto con las fases relacionadas con la implementación y experimentación. Desde este momento y hasta que se realicen todas la tareas de implementación de los tres algoritmos, etapas con ID 4, 5 y 6, la realización de la experimentación para validar la corrección del código, etapa con ID 7, y el desarrollo se la librería se harán en paralelo. La implementación del algoritmo de intersección se planifica para tener una duración de unos 60 días lectivos, finalizando a mediados de abril, seguida por la implementación del algoritmo de diferencia, con dos semanas de plazo, y por último por la implementación de la unión, con una semana de plazo. Estas tres fases son estrictamente secuenciales pues la unión se basa en la intersección y la diferencia, y la diferencia en la intersección.

Una vez finalizada la programación y la experimentación en la primera semana de mayo, comienza la última fase del desarrollo, la elaboración de la memoria (ID 8), cuya fecha límite se fija el 14 de junio del 2021 para entregar el proyecto en la primera convocatoria de este mismo curso.

2.1.2. Seguimiento

La duración real de las tareas, en líneas generales, se ha correspondido con la planificación inicial del proyecto. Esto se debe sobretodo a las mecánicas de trabajo y de seguimiento del grupo de investigación Trasgo. Todos los jueves se han realizado reuniones de trabajo donde todos los investigadores informaban de su progreso semanal. En estas reuniones, se entregaba el trabajo asignado en la reunión anterior y se recibían nuevas tareas para la semana siguiente. De esta forma, se controla el progreso del proyecto, además de recibir un feedback constante sobre el ritmo y la calidad del trabajo. A mayores, a partir de mayo se realizan reuniones los martes para revisar el estado de esta memoria.

Además de las reuniones realizadas de forma telemática vía Discord (una plataforma de videoconferencias [24]), se cuenta con una canal de texto donde se deja por escrito las tareas del proyecto que van siendo asignadas por el tutor, así como el estado de las mismas (terminadas, en proceso o inacabadas).

De esta forma, las etapas 1, 2 y 3 se realizaron en los tiempos que fueron estimados, realizando todas las tareas relacionadas con el estado del arte antes de terminar el primer cuatrimestre. Con la finalización del período ordinario de exámenes se retoma el trabajo, con el inicio de la etapa 4. Esta es la única fase del proyecto que experimenta ciertos atrasos con las entregas, debido a problemas en el código y a la aparición de errores en las salidas de los casos de prueba nuevos que señalaban a errores de validez en secciones de código realizadas tiempo atrás. Estos atrasos surgieron en concreto en la etapa 4.2. que se corresponde con la implementación del algoritmo de partición del polígono de intersección, con una demora de entre 1 y 2 semanas. Para solventar esto, se aumento el ritmo de trabajo en la semana posterior a la entrega de la tarea asociada y se logró recuperar el avance correcto de las tareas según la planificación inicial y no atrasar las posteriores etapas. Por su parte, las etapas 5, 6, 7 y 8 fueron completadas en los tiempos proyectados sin ningún tipo de percance.

2.2. Análisis de riesgos

En esta sección se van a enumerar los principales riesgos que amenazan el avance del proyecto, se va a describir el camino crítico del mismo y cuáles son las principales medidas del plan de contingencia para paliar los efectos adversos asociados a cada uno de los riesgos.

2.2.1. Riesgos

Los autores Bob Hughes y Mike Cotterell [25] definen un riesgo como “Una condición o evento incierto que, si ocurre tiene efectos positivos o negativos en los objetivos del proyecto”. Empleando la metodología de Barry Boehm [25] se han enumerado los principales riesgos que pueden afectar al proyecto en la siguiente.

En la tabla 2.2, se muestra el índice denominado como exposición al riesgo [25]. Éste es empleado para medir el daño potencial de un riesgo y es calculado como el producto de la

Descripción del riesgo	Probabilidad	Impacto	Riesgo
Desarrollo técnicamente demasiado complejo	5	4	20
Estimación no realista de los tiempos de desarrollo	6	4	24
Avería de la máquina de trabajo personal usada para el desarrollo	1	2	2
Enfermedad del programador	2	6	12
Enfermedad del tutor	2	3	6
Retrasos en el desarrollo debido a confinamientos del programador o tutores por el Covid-19	4	3	12
Resultados de la experimentación erróneos	5	3	15
Cambio en los requisitos durante el desarrollo del proyecto	1	7	7

Cuadro 2.2: Tabla de riesgos del proyecto.

probabilidad de que un riesgo se materialice por el impacto que este produce. Este valor es empleado para priorizar el uso de recurso a la hora de diseñar planes de contingencia frente a los distintos riesgos. Los valores que pueden tomar tanto la probabilidad como el impacto van de 0 a 10 y por tanto, el riesgo puede ir de 0 a 100. Como es lógico aquellos riesgos con valores más alto deberán ser los primeros en ser atendidos. Para el caso de este proyecto son el desarrollo técnico demasiado complejo y una estimación no realista de los tiempos de desarrollo.

2.2.2. Camino crítico

El camino crítico de un proyecto [26] se define como la cadena de actividades que define la duración de un proyecto y que, por ende si se experimenta un retraso en alguna de las actividades se producirá un retraso en la duración total del proyecto. Este camino crítico esta marcado en rojo en el diagrama de Gantt de la figura 2.2.

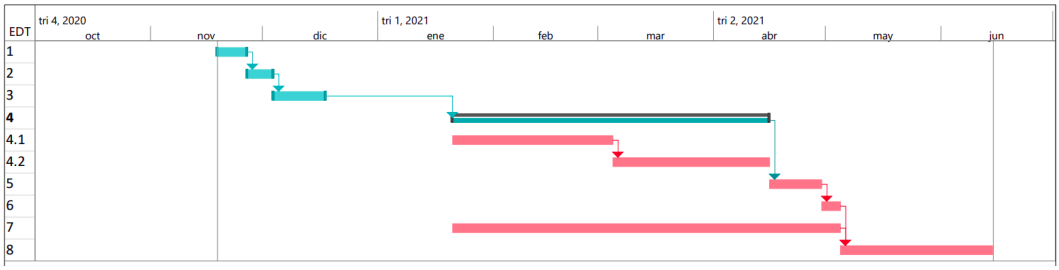

Figura 2.2: Camino crítico del proyecto.

Como se puede apreciar en el diagrama de Gantt, el camino crítico esta formado por las etapas 4 a 8 del proyecto. Esto se debe a dos razones principales:

- La secuencialidad de las tareas de implementación, en paralelo a su correspondiente experimentación, seguidas de la elaboración de este documento.
- La presencia de una período de tiempo de pausa para el proyecto entre las etapas 3 y 4, debido a la presencia de los exámenes del primer cuatrimestre.

Las tres primeras etapas, relacionadas con el estado del arte, están proyectadas para ser acabadas a mediados de diciembre y al no retomarse la actividad del proyecto hasta el segundo cuatrimestre, se cuenta con un período de un mes por el cual si se produce algún retraso en las primeras fases, este no afectará a la duración del proyecto.

Por otro lado, las etapas 4, 5, 6, 7 y 8 conforman el anteriormente definido como camino crítico, pues un retraso en cualquiera de ellas retrasaría la fecha de finalización global. La implementación de cada uno de los tres algoritmos se hace de forma secuencial, mientras la experimentación para validar su corrección se hace en paralelo. Por último, solo cuando se haya terminado tanto la implementación como la experimentación se podrá iniciar la elaboración de esta memoria. Esta secuencialidad es inherente a un proyecto de estas características donde se cuenta con un único programador.

2.2.3. Plan de contingencia

El libro "Software Project Manager" [25] define el plan de contigencia como un conjunto de acciones que se llevan a cabo para mitigar el impacto de la materialización de uno o más riesgos. En la tabla 2.3 se muestra las principales medidas propuestas para hacer frente a los riesgos descritos en la sección 2.2.1.

2.3. Presupuesto del proyecto

En esta sección se realiza un análisis del presupuesto del proyecto. El coste asociado a todo el ciclo de desarrollo se puede desglosar en dos grupos principales; la amortización de las maquinas de trabajo empleadas y las horas de trabajo del desarrollador y de los tutores. Esto es así pues no ha sido necesario la adquisición de licencias de software ni emplear máquinas del cluster del grupo de investigación Trasgo. De esta forma el presupuesto del proyecto se distribuye de la siguiente manera:

- Sueldo del desarrollador: se estima el sueldo de un ingeniero informático junior en unos 20.000 € brutos anuales. Trabajando a jornada completa (8 horas diarias) y suponiendo unos 250 días laborales al año aproximadamente, se obtiene un coste del desarrollador de 10 € la hora.

Descripción del riesgo	Plan de Contingencia
Desarrollo técnicamente demasiado complejo.	Se acotan los objetivos del proyecto.
Estimación no realista de los tiempos de desarrollo.	Se realiza la entrega del proyecto en la segunda convocatoria.
Avería de la máquina de trabajo personal usada para el desarrollo.	Se recupera el trabajo del repositorio remoto y se continua el desarrollo en las máquinas de Trasgo.
Enfermedad del programador	Se realiza el trabajo de forma telemática y entrega del proyecto en segunda convocatoria.
Enfermedad del tutor.	Se realizan las consultas sobre el desarrollo del proyecto al resto de tutores u otros investigadores del grupo Trasgo.
Retrasos en el desarrollo debido a confinamientos del programador o tutores por el Covid-19.	Se realiza trabajo y seguimiento telemático vía discord y con repositorios remotos.
Resultados de la experimentación erróneos.	Se realiza la búsqueda del origen de los errores mediante las herramientas valgrind y GDB. En caso de no solucionarlos declararlos como trabajo futuro.
Cambio en los requisitos durante el desarrollo del proyecto.	Se aumentan las horas de trabajo semanales para alcanzar los nuevos objetivos.

Cuadro 2.3: Plan de contingencia del proyecto.

- Sueldo del tutor personal docente investigador predoctoral: se estima el sueldo de un investigador doctorando en unos 14000 € brutos anuales. Trabajando a jornada completa (8 horas diarias) y suponiendo unos 250 días laborales al año aproximadamente, se obtiene un coste del desarrollador de 7 € la hora.
- Sueldo del tutor personal docente investigador: se estima el sueldo de un investigador doctorando en unos 28000 € brutos anuales. Trabajando a jornada completa (8 horas diarias) y suponiendo unos 250 días laborales al año aproximadamente, se obtiene un coste del desarrollador de 14 € la hora.
- Coste de la máquina de desarrollado: la máquina utilizada se trata de un Lenovo Thinkpad de 16 GB de RAM, un procesador Intel Core i7 y una memoria gráfica GeForce MX250. Tuvo un coste de 1400 € y con estimación de vida útil de 5 años. El coste asociado a la amortización es de 0,31 €/hora.

Las horas de utilización de cada recurso se proyectan de la siguiente forma:

- Desarrollador: 300 horas asociadas a los 12 créditos de la asignatura del Trabajo de Fin de Grado.
- Tutora personal docente investigador predoctoral: el desarrollo de este proyecto se enmarca dentro de uno de los proyectos de investigación del grupo Trasgo y la tutora

académica María Inmaculada Santamaría es la principal investigadora en el mismo. Se estima que invertirá unas 100 horas totales.

- Tutor personal docente investigador: con el tutor académico Yuri Torres se realizan 8 reuniones de una hora a partir del 4 de mayo con motivo de revisión de esta memoria. A mayores, todos los jueves desde el inicio del proyecto se realizan reuniones de seguimiento de una duración aproximada de media hora en las que participan Yuri Torres y Arturo González. El total de horas empleadas por el personal docente investigador en este proyecto se estima en 23h.
- Máquina del desarrollador: se utiliza durante la totalidad del proyecto, por ende, 300 horas.

Bajo estas condiciones, el coste de trabajo del personal y de uso de las máquinas en el proyecto viene descrito en las tablas 2.4 y 2.5.

Personal	**Coste/hora (€)**	**Horas**	**Total**
Alumno	10	300	3000
Tutora PDI predoctoral	7	100	700
Tutor PDI 1	14	15	210
Tutor PDI 2	14	8	112
Total			4022

Cuadro 2.4: Coste de trabajo de personal del proyecto.

Máquina	**Precio (€)**	**Amortización (años)**	**Coste/hora (€)**	**Horas**	**Total (€)**
Del desarrollador	14000	5	0.31	300	93
Total					93

Cuadro 2.5: Coste de uso de las máquinas en el proyecto.

En la tabla 2.6 se indica cual es el coste total del proyecto, calculado a partir del coste de las máquinas y de las horas de trabajo del personal.

Actividad	**Coste (€)**
Horas de trabajo del personal	4022
Uso de las máquinas	93
Total	**4115**

Cuadro 2.6: Coste total del proyecto.

Capítulo 3

Análisis y diseño

En este capítulo se introducen los siguientes aspectos:

- Un análisis sobre la funcionalidad de Hitmap que va a ser ampliada.
- La funcionalidad de los algoritmos seleccionados (intersección, unión y diferencia); una descripción genérica.
- Los fundamentos matemáticos de los diferentes algoritmos seleccionados.
- El diseño de cada uno de los tres algoritmos.

3.1. Análisis

El objetivo de este proyecto es es ampliar la funcionalidad de la librería Hitmap del grupo de investigación de Trasgo, para permitir el particionado triangular del espacio de memoria, también denominado como la triangularización de espacio de datos.

Tal y como se ha descrito en la sección 1.2.3., Hitmap es una librería diseñada para desacoplar el patrón de comunicación del particionado de datos, mediante el uso de expresiones abstractas que se adaptan automáticamente en tiempo de ejecución dependiendo de la partición finalmente utilizada. Para lograr esto, la librería trabaja con las siguientes estructuras [14]:

- **Signature:** Una *Signature* S es una tupla de tres elementos enteros que representa un subespacio de índices de una array un en dominio de una dimensión. De esta forma, la definición de una signature es la siguiente:

  $S \in Signature = (begin : end : stride)$
  $Card(s \in Signature) = [(s.end - s.begin)/s.stride]$

- **Shape:** Un *Shape* h es una tupla de n signatures. Representa la selección de un subespacio de índices en un dominio multidimensional, formando un n-paralelootopo [27] (generalización de n dimensiones de una paralelepípedo).

  $$S \in Shape = (S_0, S_1, S_2, ..., S_{n-1})$$
  $$Card(s \in Shape) = \prod_{i=0}^{n-1} Card(S_i)$$

- **Tile:** Una *Tile* es un array de n dimensiones. Su dominio está definido por un shape, y tiene un número de elementos de un tipo dado, dependiendo del lenguaje de programación empleado.

  $$Tile_{h \in Shape} : (S_0 \times S_1 \times S_2 \times ... \times S_{n-1}) \rightarrow < type >$$

A partir de estas estructuras Hitmap ofrece las siguientes funcionalidades [14]:

- **Tiling.** Conjunto de funciones para la definición y manipulación de arrays y tiles. Pueden ser empleadas independientemente de las otras funciones para mejorar la localidad en código secuencial así como para generar distribuciones de datos manualmente para ejecuciones en paralelo.

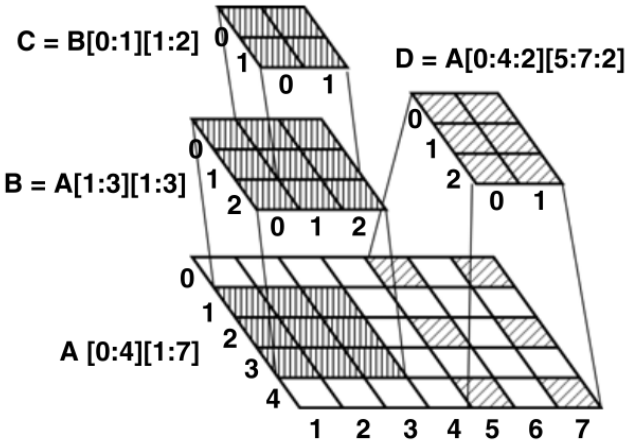

Figura 3.1: Creación de tiles a partir de arrays (figura extraida de [14]).

- **Mapeado.** Conjunto de funciones para la distribución y particionado automático de datos en función de la topología virtual seleccionada. Estas funciones están orientadas a la distribución de datos y tareas en entornos paralelos.

  Estas funciones toman como entrada una topología virtual y un Layout [14] y generan como salida una estructura de datos que modela el rango de Tiles que debe ser creada, el mappeo entre tiles y los procesadores virtuales e información sobre los procesadores vecinos de cada procesador.

- **Comunicación.** Conjunto de funciones para la creación de patrones reusables de comunicación para tiles distribuidos jerárquicamente. Estas funciones son una abstracción de un modelo de paso de mensajes para comunicar tiles entre procesadores virtuales y puede ser usado para crear mapeados dependientes de los patrones de comunicación.

La arquitectura software de las clases de la librería Hitmap está descrita en la imagen 3.2. Las clases que aportan la funcionalidad de tiling son las que presenta fondo blanco, las

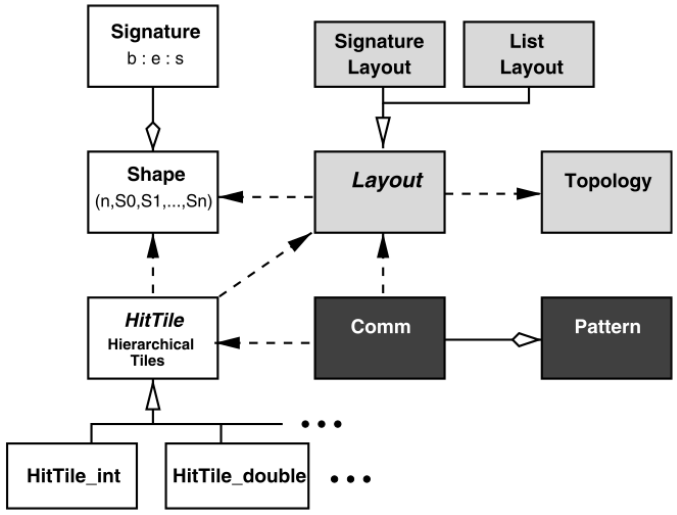

Figura 3.2: Diagrama UML de la arquitectura de la librería Hitmap (Figura extraída de [14]).

de mapeado las de fondo gris y las de comunicación las tienen el fondo negro.

3.2. Funcionalidad de los algoritmos

En este sección se va a describir la funcionalidad a alto nivel de cada uno de los tres algoritmos, describiendo sus posibles entradas, su comportamiento y la salida esperada. Dicha funcionalidad está descrita en mayor profundidad en el Technical Report [21] desarrollado por María Inmaculada Santamaría Valenzuela.

3.2.1. Intersección

- **Objetivo del algoritmo:** Dados dos Shapes de dimensión 2, Sa y Sb, devolver un array de Shapes de dimensión 2, cuyos elementos componen la intersección de Sa y Sb.
- **Entradas:** Dos Shapes de 2 dimensiones, de tipo triangulo o rectángulo. De esta forma, la intersección entre Sa y Sb, puede ser de dos rectángulos (opción ya implementada por Hitmap), dos triángulos o de un triángulo y un rectángulo.

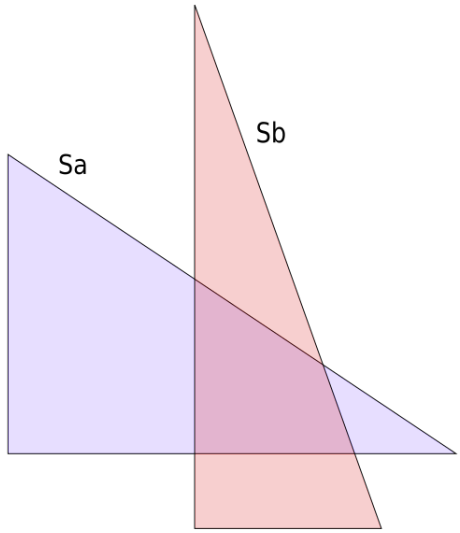

Figura 3.3: Caso de prueba 1 de la intersección.

- **Salida:** Un array de Shapes de 2 dimensiones, compuesto por los polígonos que conforman la partición del polígono de intersección. En la figura 3.4 se puede apreciar en rojo los 3 Shapes que conforman dicha partición.

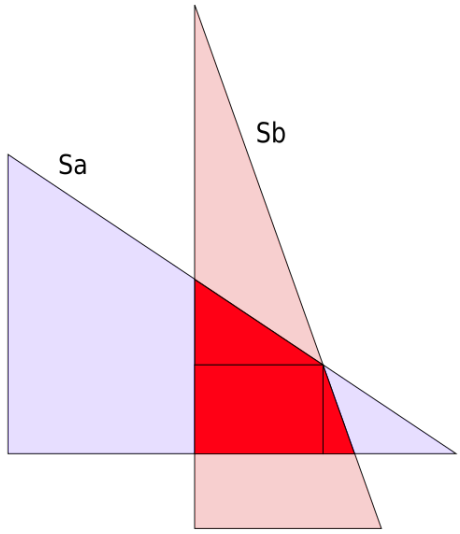

Figura 3.4: Partición del polígono de intersección del caso de prueba 1.

- **Consideraciones:**
  - La salida será una array de Shapes. Estos Shapes solo podrán ser de aquellos

tipos modelados por la librería Hitmap [14] y su extensión, es decir, rectángulos o triángulos.

- Las *Signatures* son tuplas de tres elementos enteros que representan un subconjunto de índices pertenecientes a $\mathbb{Z}^2$.
- Las intersecciones entre los lados de los Shapes de entrada dan lugar a los vértices del polígono de intersección con coordenadas en $\mathbb{R}^2$ y no en $\mathbb{Z}^2$. Esto obliga al algoritmo a realizar un reajuste a coordenadas enteras de los polígonos para poder ser modelados mediante las estructuras Signature y Shape.
- El reajuste de los polígonos de la partición del polígono de intersección da lugar a puntos aislados con coordenadas enteras que pertenecen al polígono de intersección pero que al reajustarse ya no se encuentran dentro de los Shapes reajustados. Estos puntos son denominados outsiders y hay que añadirlos al array de Shapes de salida.
- El conjunto de polígonos que conforman la partición del polígono de intersección debe ser disjunto, es decir, un índice no puede pertenecer a más de un polígono. Esto asegura que una posición de memoria o dato es procesada única y exclusivamente por una unidad de computo concreta.
- Los outsiders pueden ser modelados mediante Shapes en forma de triángulos o rectángulos de dimensión 0. Para ello, los campos begin y end, de las signatures de los polígonos deben tener el mismo valor.

Como consecuencia de estas consideraciones obtenemos que la salida de la intersección para el caso de prueba 1, indicado en la figura 3.3, es la siguiente: Los Shapes coloreados en rojo

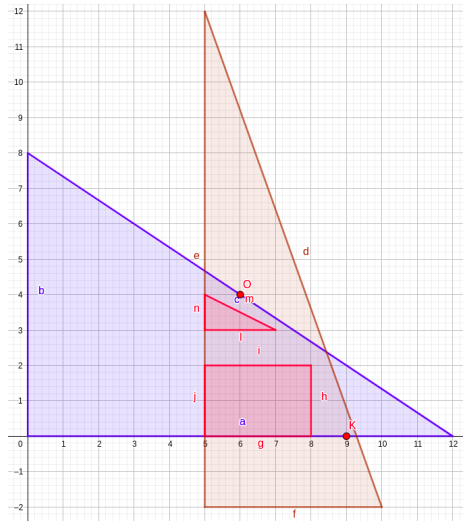

Figura 3.5: Salida del caso de prueba 1 para el algoritmo de intersección.

se corresponde con los elementos que componen el array de salida del algoritmo. Como se aprecia en la figura 3.5 se tratan de un rectángulo, un triángulo y dos puntos.

3.2.2. Diferencia

- **Objetivo del algoritmo:** Dados 2 Shapes de dimensión 2, Sa y Sb, devolver un array de Shapes de dimensión 2, cuyos elementos componen la diferencia de Sa menos Sb.
- **Entradas:** Dos Shapes de 2 dimensiones, de tipo triangulo o rectángulo. De esta forma, la diferencia entre Sa y Sb, puede ser de dos rectángulos, dos triángulos o de un triángulo y un rectángulo.

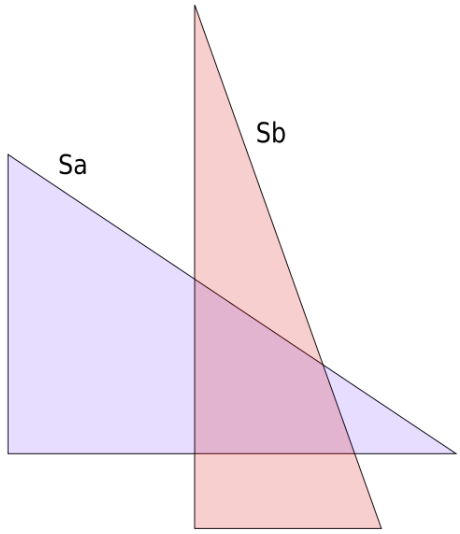

Figura 3.6: Caso de prueba 1 de la diferencia.

- **Salida:** Un array de Shapes de 2 dimensiones, compuesto por los polígonos que conforman la partición de los polígonos de la diferencia. En la figura 3.7 se puede apreciar en rojo los Shapes que conforman dicha partición.
- **Consideraciones:**
  - La salida será una array de Shapes. Estos Shapes solo podrán ser de aquellos tipos modelados por la librería Hitmap y su extensión, es decir, rectángulos o triángulos.
  - Las *Signatures* son tuplas de tres elementos enteros que representan un subconjunto de índices pertencientes a $\mathbb{Z}^2$.
  - Las intersecciones entre los lados de los Shapes de entrada dan lugar a los vértices del polígono de intersección con coordenadas en $\mathbb{R}^2$ y no en $\mathbb{Z}^2$. Esto obliga al algoritmo a realizar un reajuste a coordenadas enteras de los polígonos para poder ser modelados mediante las estructuras Signature y Shape.
  - El reajuste de los polígonos de la partición de los polígonos de la diferencia da lugar a puntos aislados con coordenadas enteras que pertenecen a la diferencia pero que al reajustarse ya no se encuentran dentro de los Shapes reajustados. Estos puntos son denominados outsiders y hay que añadirlos al array de Shapes de salida.

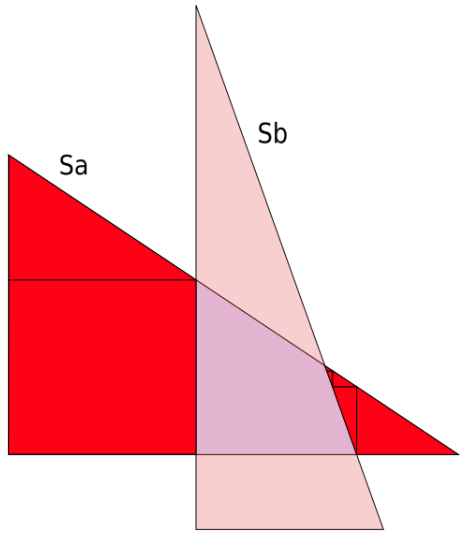

Figura 3.7: Partición de los polígonos de la diferencia del caso de prueba 1.

- El conjunto de polígonos que conforman la partición del polígono de intersección debe ser disjunto, es decir, un índice no puede pertenecer a más de un polígono. Esto asegura que una posición de memoria o dato es procesada única y exclusivamente por una unidad de computo concreta.
- Los outsiders pueden ser modelados mediante Shapes en forma de triángulos o rectángulos de dimensión 0. Para ello, los campos begin y end, de las signatures de los polígonos deben tener el mismo valor.
- Los polígonos de la partición de la diferencia no pueden tener puntos comunes con el Shape Sb, ni siquiera en su perímetro. Es por ello que será necesaria una fase extra de reajuste con respecto a la intersección para asegurar que ningún punto de la salida coincide con alguno de los lados de Sb.

Como consecuencia de estas consideraciones obtenemos que la salida de la diferencia para el caso de prueba 1, indicado en la 3.6, es la siguiente:

3.2.3. Unión

- **Objetivo del algoritmo:** Dados dos Shapes de dimensión 2, Sa y Sb, devolver un array de Shapes de dimensión 2, cuyos elementos componen la unión de Sa y Sb.
- **Entradas:** Dos Shapes de 2 dimensiones, de tipo triangulo o rectángulo. De esta forma, la unión entre Sa y Sb, puede ser de dos rectángulos, dos triángulos o de un triángulo y un rectángulo.

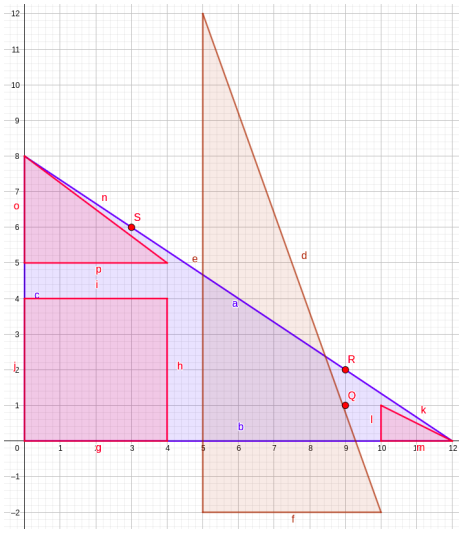

Figura 3.8: Salida del caso de prueba 1 para el algoritmo de la diferencia.

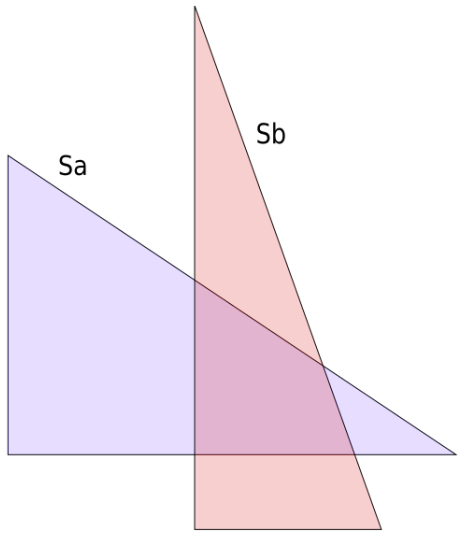

Figura 3.9: Caso de prueba 1 de la unión.

- **Salida:** Un array de Shapes de 2 dimensiones, compuesto por los polígonos que conforman la partición de los polígonos de la unión. En la figura 3.7 se puede apreciar en rojo los Shapes que conforman dicha partición.
- **Consideraciones:**
  - La salida es una array de Shapes. Estos Shapes solo pueden ser de aquellos tipos

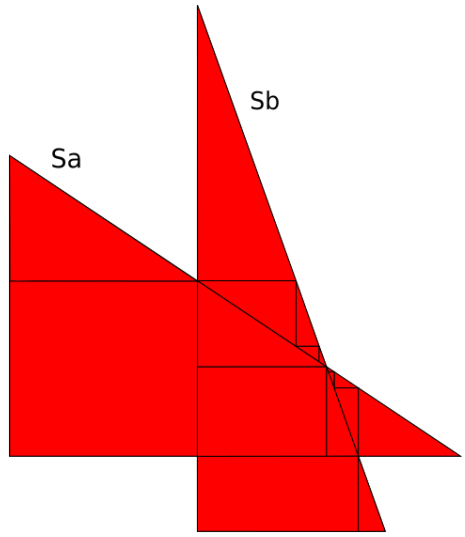

Figura 3.10: Partición del polígono de la unión del caso de prueba 1.

modelados por la librería Hitmap y su extensión, es decir, polígonos rectangulares o triangulares.

- Las *Signatures* son tuplas de tres elementos enteros que representan un subconjunto de índices pertencientes a $\mathbb{Z}^2$.
- Las intersecciones entre los lados de los Shapes de entrada dan lugar a los vértices del polígono de intersección con coordenadas en $\mathbb{R}^2$ y no en $\mathbb{Z}^2$. Esto obliga al algoritmo a realizar un reajuste a coordenadas enteras de los polígonos para poder ser modelados mediante las estructuras Signature y Shape.
- El reajuste de los polígonos de la partición de los polígonos de la diferencia da lugar a puntos aislados con coordenadas enteras que pertenecen a la diferencia pero que al reajustarse ya no se encuentran dentro de los Shapes reajustados. Estos puntos son denominados outsiders y hay que añadirlos al array de Shapes de salida.
- El conjunto de polígonos que conforman la partición del polígono de intersección debe ser disjunto, es decir, un índice no puede pertenecer a más de un polígono. Esto asegura que una posición de memoria o dato es procesada única y exclusivamente por una unidad de computo concreta.
- Los outsiders pueden ser modelados mediante Shapes en forma de triángulos o rectángulos de dimensión 0. Para ello, los campos begin y end, de las signatures de los polígonos deben tener el mismo valor.
- La salida del algoritmo de unión de dos Shapes, Sa y Sb, es equivalente a las salidas de la diferencia de Sa menos Sb, la salida de Sb menos Sa y la salida de la intersección de Sa y Sb, almacenadas en un único array.

Como consecuencia de estas consideraciones obtenemos que la salida de la unión para el caso

de prueba 1, indicado en la 3.9, es la siguiente:

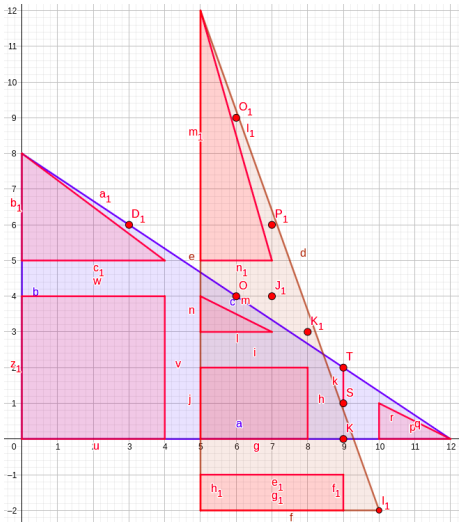

Figura 3.11: Salida del caso de prueba 1 para el algoritmo de unión.

3.3. Requisitos funcionales y no funcionales

Resultado de las características y funcionalidades ofrecidas por la librería Hitmap, así como de los objetivos presentes en la extensión de la misma a desarrollar en este proyecto, se establecen un conjunto de requisitos que debe cumplir el nuevo software. Estos requisitos se categorizan en dos grupos principales, funcionales y no funcionales.

Los requisitos funcionales [28] son declaraciones de los servicios y funcionalidades que ofrecerá el software a desarrollar y de la forma en que éste se comportará a entradas particulares. La extensión de la librería Hitmap a desarrollar durante este proyecto presenta los siguientes requisitos funcionales:

- **RF01:** La librería debe modelar particiones triangulares del espacio de memoria.

  Como se ha descrito en la sección 1.2.4, el principal objetivo de la extensión de la librería Hitmap que se va a desarrollar es la posibilidad de admitir triángulos como unidad de división del particionado del espacio de datos. Esto permite a la librería Hitmap adapatarse mejor a la geometría del problema de computo.

- **RF02:** La librería debe calcular la intersección entre dos shapes.

Para poder trabajar con las particiones triangulares de memoria es necesario poder obtener la intersección entre dos shapes (estructura definida por Hitmap) y obtener su correspondiente partición en polígonos manejados por la librería y su correspondiente extensión.

- **RF03:** La librería debe calcular la diferencia entre dos shapes.

  Para poder trabajar con las particiones triangulares de memoria es necesario poder obtener la diferencia entre dos shapes y obtener su correspondiente partición en polígonos manejados por la librería y su correspondiente extensión.

- **RF04:** La librería debe calcular la unión entre dos shapes.

  Para poder trabajar con las particiones triangulares de memoria es necesario poder obtener la unión entre dos shapes y obtener su correspondiente partición en polígonos manejados por la librería y su correspondiente extensión.

Los requisitos no funcionales [28] son aquellos requerimientos que no se refieren directamente a las funciones específicas que ofrece el software, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento, así como las restricciones del sistema. La extensión de la librería Hitmap a desarrollar durante este proyecto presenta los siguientes requisitos no funcionales:

- **RNF01:** Los shapes manejados por la librería podrán representar rectángulos, triángulos, segmentos o puntos.

  Como se describe en la sección 3.1, la estructuras de la librería Hitmap son capaces de modelar rectángulos de lados paralelos a los ejes de coordenadas, así como puntos y segmentos. Para lograr la triangularización del espacio de datos es necesario extender la estructura HitSigExtShape para que pueda modelar también triángulos.

- **RNF02:** Los cálculos realizados por la librería para los algoritmos de intersección, diferencia y unión deben ser correctos.

  Los cálculos realizados por dichos algoritmos serán empleados para manejar el particionado de memoria de un espacio de datos. Si estas operaciones son realizadas de forma incorrecta o con salidas distintas a las esperadas puede degenerar en errores en la gestión de memoria y de procesos en las aplicaciones que utilicen la librería Hitmap en el ámbito de la computación paralela heterogénea.

- **RNF03:** Las particiones realizadas para las salidas de los algoritmos deben ser disjuntas y todo índice de memoria de la salida debe pertenecer a una partición.

  El objetivo del particionado es que todo índice sea gestionado únicamente por un procesador, con el objetivo de evitar cálculos redundantes, en caso de un índice que pertenece a más de un partición, y pérdida de información, en caso de que un índice no pertenezca a ninguna partición. Para ello, se debe asegurar que las salidas de los algoritmos sean disjuntas y recojan todos los índices necesarios.

- **RNF05:** El software debe integrarse correctamente en la arquitectura de la librería Hitmap.

  La librería Hitmap cuenta con una estructura de directorios de cierta complejidad y que responde a unos criterios de diseño propuestos por el grupo de investigación Trasgo. Es por ello que los ficheros creados para extender la librería deben ser incluidos correctamente en la arquitectura de la librería.

- **RNF04:** El lenguaje de programación empleado para el desarrollo debe ser C.

  El lenguaje de programación empleado debe ser el mismo que ha sido empleado por el grupo de Investigación trasgo para la implementación de la librería Hitmap. Es por ello que el desarrollo se realizará en C.

3.4. Fundamentos matemáticos de los algoritmos

En esta sección se enumeran los fundamentos y teoremas matemáticos en los que se basan los algoritmos de intersección, diferencia y unión.

Existen una serie de fundamentos matemáticos comunes a los tres algoritmo que son empleados como base para la construcción de sus respectivas operaciones. Son ejemplos de esto la teoría de conjuntos y de partición.

A lo largo de todo el proyecto se trabajan con conjuntos de índices que representan posiciones de memoria con la particularidad de que su geometría viene definida por la estructura Shape de la librería Hitmap. Las operaciones de intersección, diferencia y unión se realizarán sobre dos conjuntos obteniendo un subconjunto o un nuevo conjunto, en función del algoritmo. Es por ello que se procede a definir lo que es un conjunto:

- **Conjunto** [29] es una colección bien definida de objetos, es decir, está definida de forma que para un objeto x cualquiera, podamos determinar si x pertenece o no al conjunto. Los objetos que pertenecen al conjunto se llaman elementos o miembros. Denotaremos los conjuntos por letras mayúsculas, tales como A o X; si a es un elemento del conjunto A, escribimos a $\in A$.

Una particularidad de los algoritmos implementados en esta extensión de Hitmap es que los conjuntos de salida se almacenan en un array de Shapes. Estos Shapes son triángulos

o rectángulos que conforman la partición del conjunto inicial, habilitando la posibilidad de asignar cada partición a un procesador virtual. Es por ello que se procede a definir lo que es una partición:

- **Partición** P de un conjunto X, es una colección de conjuntos no vacíos X_1, X_2, ... tales que $X_i \bigcap X_j = \emptyset$ para i $\neq$ j y $\bigcup_k X_k$ = X. Sea una relación de equivalencia en un conjunto X y sea $x \in X$. Entonces $[x] = \{y \in X : y - x\}$ se llama clase de equivalencia de x.

3.4.1. Intersección

Según la teoría de conjuntos, la intersección de dos conjuntos A y B, no vacíos, se define como [29]:

$$A \bigcap B = \{x : x \in A \text{ y } x \in B\}$$

Teorema: La intersección de un polígono convexo de n lados con otro convexo de m lados es un polígono convexo de cómo mucho $n + m$ vértices [30].

Como consecuencia del teorema anterior, la intersección calculada por la librería Hitmap obtiene como salida un polígono de intersección con un máximo de 8 vértices (intersección entre dos rectángulos). El polígono de intersección de dos Shapes, Sa y Sb, viene determinado por los siguientes vértices:

- Los puntos resultantes de la intersección entre los lados de Sa y Sb.
- Los vértices de Sa que pertenecen a Sb.
- Los vértices de Sb que pertenecen a Sa.

Para la determinación de aquellos vértices obtenidos a través de la intersección de los lados de los Shapes Sa y Sb, se emplean las ecuaciones de las rectas que estos lados determinan [31]:

$$a_1x + b_1y = c_1 \qquad a_2x + b_2y = c_2$$

y se obtienen, a partir de la regla de Cramer o sustituyendo una variable, las coordenadas del punto de intersección $(x_s, y_s)(x_s, y_s)$:

$$x_s = \frac{c_1b_2 - c_2b_1}{a_1b_2 - a_2b_1}, \quad y_s = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1}$$

(Si $a_1b_2 - a_2b_1 = 0$ las líneas son paralelas y estas fórmulas no se pueden usar porque implican dividir por 0).

A mayores es necesario tener en cuenta una serie de consideraciones:

1. La intersección de las dos figuras puede ser un punto o una línea. Esta salida puede ser modelada por las estructuras de Hitmap, a través de Shapes de dimensión 0 y 1, respectivamente.

2. Los polígonos pueden simplemente no interseccionar y estár separados. El algoritmo devolverá un conjunto vacío.

3. Un polígono puede estar contenido dentro del otro. Como resultado de la intersección de ambos polígonos se obtendrá el polígono que está siendo contenido.

3.4.2. Diferencia

Según la teoría de conjuntos, la diferencia para dos conjuntos, A y B, no vacíos, se define como [29]:

$$A - B = \{x : x \notin A \text{ y } x \in B\}$$

Existen una serie de propiedades de la teoría de conjuntos que debe cumplir el algoritmo de la diferencia para su correcto funcionamiento:

- La diferencia de dos polígonos, Sa y Sb, teniendo que Sb es un conjunto vacío, tiene como resultado el propio Sa. Esto se debe a la propiedad del elemento neutro:

$$A - \emptyset = A$$

- La diferencia de un polígono Sa menos él mismo es el conjunto vacío.

$$A - A = \emptyset$$

- La diferencia de dos polígonos, Sa y Sb, teniendo que ningún punto de Sb pertenece Sa, es decir, son dos conjuntos disjuntos, tiene como resultado Sa.

$$\text{A - B} = \text{A} \leftrightarrow A \bigcap B = \emptyset$$

- La diferencia de dos polígonos, Sa y Sb, es el conjunto vacío si Sa está contenido en Sb, es decir, Sa se trata de un subconjunto de Sb.

$$\text{A - B} = \emptyset \leftrightarrow A \subseteq B$$

3.4.3. Unión

Según la teoría de conjuntos la unión [29] de dos conjuntos, A y B, no vacíos, se define como:

$$A \bigcup B = \{x : x \in A \text{ o } x \in B\}$$

Existen una serie de propiedades de la teoría de conjuntos que debe cumplir el algoritmo de la diferencia para su correcto funcionamiento:

- La unión de un polígono, Sa, consigo mismo es el propio Sa. Esto se debe a la propiedad de Idempotencia:

$$A \cup A = A$$

- La unión de dos polígonos, Sa y Sb, siendo Sb un polígono contenido por Sa tiene como resultado el propio Sa.

$$B \subseteq A \rightarrow A \cup B = A$$

- La unión de un polígono Sa con un conjunto vacío es el propio Sa. Esto se debe a la propiedad del elemento neutro:

$$A \cup \emptyset = A$$

3.5. Diseño

En esta sección se describen las principales decisiones de diseño tomadas para el desarrollo de la extensión de la librería Hitmap y en particular para cada uno de los tres algoritmos (intersección, diferencia y unión).

Previo a la realización y desarrollo de este proyecto, la librería contaba con las estructuras HitSig y HitSigShape para modelar las particiones rectangulares del espacio de datos. La estructura HitSigShape cuenta con un array con tantos elementos como dimensiones del espacio tenga el cuerpo modelado.

Para el caso del rectángulo, HitSigShape está compuesto por dos HitSig, que indican el índice de inicio y de final para cada dimensión del espacio. La estructura HitSig debe construirse de forma que su campo begin sea menor que end.

```
typedef struct {
        int begin;       /**< The begin index of the dimension */
        int end;         /**< The end index of the dimension */
        int stride; /**< The stride for regular sparse domains */
} HitSig;
```

```
typedef struct {
        /** @privatesection */
        int numDims; /**< Number of dimensions.**/
        HitSig  sig[HIT_MAXDIMS]; /**< HitSig objects to define each dimension
            .*/
} HitSigShape;
```

Estas dos estructuras deberán ser extendidas para poder modelar particiones triangulares. Como resultado, se crean las estructuras HitSigExt y HitSigExtShape. La primera cuenta con los mismos campos, pero con la diferencia de que el campo begin puede ser mayor que end. Como se explicará más adelante, esta característica permitirá modelar los distintos tipos de triángulo que maneja la librería.

```
typedef struct {
        int begin;       /**< The begin index of the dimension */
        int end;         /**< The end index of the dimension */
        int stride; /**< The stride for regular sparse domains */
} HitSigExt;
```

La estructura HitSigShape ha sido extendida con el campo entero *type*, que puede tomar el valor 0 o 1 dependiendo de si la Shape reprensenta un rectángulo o un triángulo, respectivamente.

```
typedef struct {
    /** @privatesection **/
        int numDims;     /**< Number of dimensions */
        int type;        /**<0 if rectangular shape, 1 if triangular shape */
        HitSigExt sig[HIT\_MAXDIMS]; /**< The stride for extended sparse
            domains */
} HitSigExtShape;
```

Los triángulos modelados por la extensión propuesta en este proyecto presentan una serie de características:

- Son triángulos rectángulos de base y altura paralelos a los ejes de coordenadas.
- Esos triángulos pueden ser de cuatro tipos: T1, T2, T3 y T4. Este tipo viene determinado por la orientación de la diagonal del rectángulo con respecto a los ejes de coordenadas.
- El modelado de cada uno de los cuatro tipos de triángulos se realiza a través de los campos begin y end de las estructuras HitSigExt. El uso de estos valores para modelar cada triángulo viene expresado en la figura 3.12.

3.5.1. Intersección

El algoritmo de intersección se ha diseñado entorno a dos fases principales: la **construcción del polígono de intersección** y la **partición del polígono del intersección**. El objetivo final de estas dos fases es la determinación del polígono de la intersección de los Shapes de entrada a través de sus vértices y representar este polígono a través de un array de Shapes que conforman la partición del mismo.

Como consecuencia, el esqueleto principal del algoritmo de intersección para R^2 de dos figuras, (S_a y S_b), pudiendo ser dos triángulos, dos rectángulos o un triángulo y un rectángulo, es de la forma:

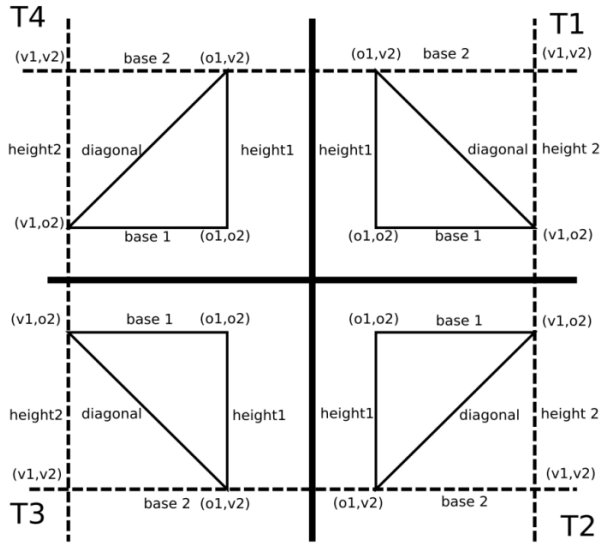

Figura 3.12: Representación de triangulos a través de la estructura HitSigExtShape (Figura extrída de [21]).

1. **Construcción del polígono de intersección**. Objetivo: Obtener un array con los vértices que conforman el polígono de intersección. Estos vértices se obtienen a través de las intersecciones entre los lados de los polígonos de entrada.

   a) **Si ambas figuras son rectángulos:**

      1. La intersección ya está definida en la librería HitMap.
         Función: HitShape hit_shapeIntersect(HitShape sh1, HitShape sh2) en hit_shape.c

   b) **Si Sa es un triángulo y Sb es un rectángulo:**

      1. Intersección de la altura de S_b con la base y diagonal de S_a
      2. Intersección de la base de S_b con la altura y diagonal de S_a

   c) **Si ambas figuras son triángulos:**

      1. Intersección de la altura de S_a con la base y diagonal de S_b
      2. Intersección de la base de S_a con la altura y diagonal de S_b
      3. Intersección de la diagonal de S_a con la altura, base y diagonal de S_b

   A mayores será necesario añadir al polígono de intersección:

   - Vértices de S_a que pertenecen a S_b
   - Vértices de S_b que pertenecen a S_a

2. **Partición del polígono de intersección**. Objetivo: Dado un array con los vértices del polígono de intersección, obtener un array con los polígonos generados por la partición (HitSigExtShapes).

1. **Búsqueda de los triángulos de la partición.** Objetivo: Dado un array ordenado de vértices, obtener los vértices que determinan los polígonos de la partición de la intersección. La búsqueda de estos triángulos se realiza en dos fases; en ambas el objetivo es la búsqueda de los puntos interiores al polígono de intersección que determinan los triángulos de la partición del mismo.
   a. **Búsqueda horizontal**
   b. **Búsqueda vertical**
2. **Hallar puntos y segmentos aislados.** Objetivo: Búsqueda de aquellos puntos y segmentos que forma parte de la intersección de Sa y Sb, pero que no han sido incluidos en ningún triángulo de la partición y añadirlos al array de salida del algoritmo modelados mediante la estructura Shape.
3. **Reajuste de los polígonos de intersección.** Objetivo: modificar los triángulos hallados durante la búsqueda de triángulos de la partición para que tenga coordenadas enteras y puedan ser modelados mediante la estructura Shape. Si dos triángulos comparten diagonal así como los vértices que la delimitan serán fusionados para formar un rectángulo y mejorar la eficiencia del particionado.

La fase de búsqueda de los triángulos de partición presenta una especial complejidad, debido sobretodo al flujo del algoritmo. Esta complejidad en el flujo es resultado de la gran cantidad de condiciones que debe cumplir un punto encontrado durante los avances de las búsquedas para que sea almacenado, así como el segmento que este determina con el punto del que proviene el avance. Con el objetivo de reflejar el funcionamiento de las búsquedas horizontal y vertical, en las figuras 3.13 y 3.14 se muestran sus diagramas de flujo:

3.5.2. Diferencia

El algoritmo de la diferencia está altamente basado en el algoritmo de la intersección y alrededor del 70 % de la funcionalidad de la intersección puede ser reutilizada.

Partiendo del algoritmo de la intersección, existen una serie de diferencias en el algoritmo de la diferencia:

1. Los polígonos de la diferencia de dos Shapes, Sa y Sb, vienen determinados por los vértices del polígono de la intersección, y a mayores, los vértices de los Shapes de entrada.

2. Las búsquedas de triángulos para la partición del polígono de intersección se realizarán de forma que se alcancen puntos que pertenezcan a Sa, pero no a la intersección de Sa y Sb. De esta forma, se obtiene la partición de la diferencia.

3. En la fase de ajuste, será necesario realizar un ajuste extra con respecto a la intersección. Este hecho se debe a la necesidad de separar las particiones obtenidas durante las búsquedas del polígono de Sb para que no se compartan puntos con el esqueleto de Sb.

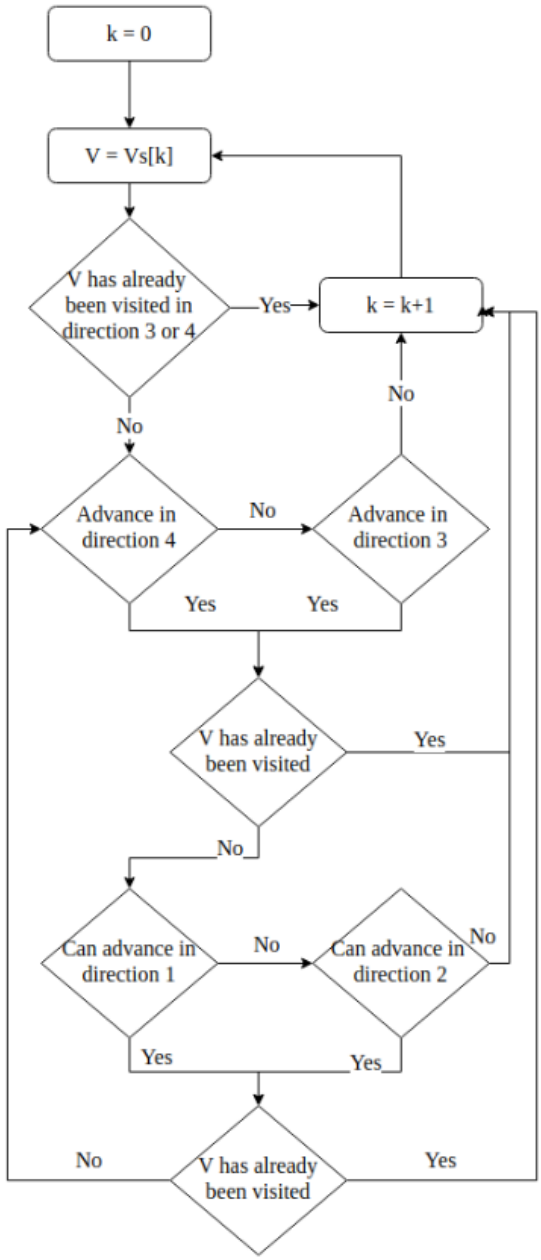

Figura 3.13: Diagrama de flujo de la búsqueda horizontal (figura extraída de [32]).

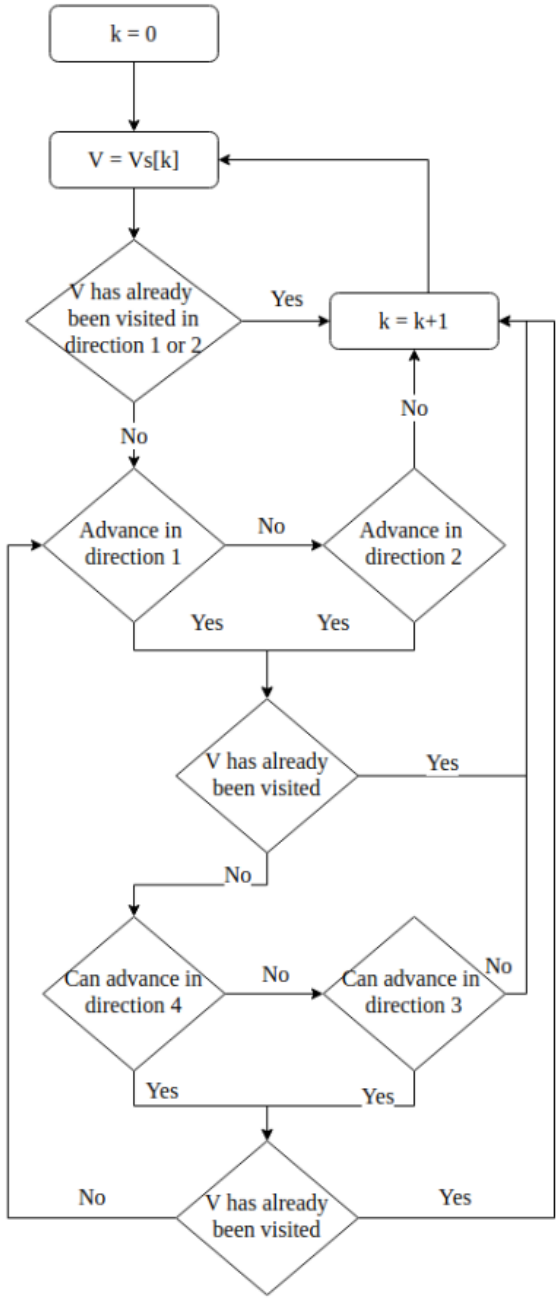

Figura 3.14: Diagrama de flujo de la búsqueda vertical (figura extraída de [32]).

3.5.3. Unión

El algoritmo de la unión está basado en los dos algoritmos anteriores. Este hecho se debe a que matemáticamente, la unión de dos conjuntos A y B está formada por:

- Los elementos que conforman la intersección de A y B.
- Los elementos que conforman la diferencia de A menos B.
- Los elementos que conforman la diferencia de B menos A.

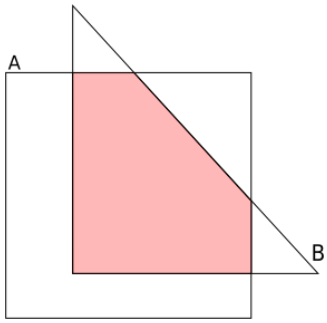

Figura 3.15: Intersección de A y B.

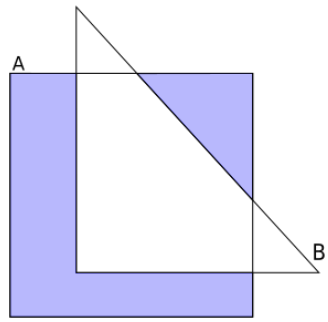

Figura 3.16: Diferencia de A menos B.

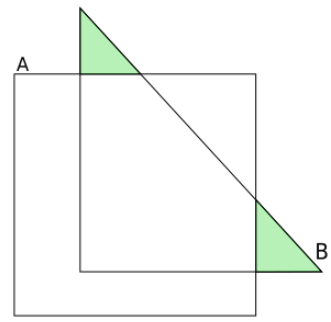

Figura 3.17: Diferencia de B menos A.

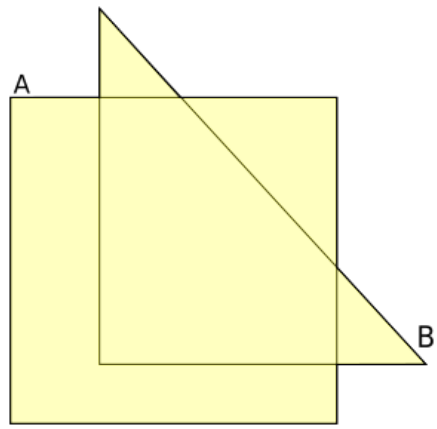

Figura 3.18: Unión de A y B.

Figura 3.19: Equivalencia entre la unión y la suma de la intersección y las diferencias.

Como resultado de la equivalencia entre la unión de dos conjuntos A y B y la suma de la intersección entre A y B, la diferencia de A menos B y la diferencia de B menos A, se decide que el algoritmo de unión de dos *Shapes*, Sa y Sb sigue el siguiente esuqema:

1. **Intersección de *Sa* y *Sb*:** Cálculo de la intersección entre ambos Shapes mediante la llamada a la función del algoritmo de intersección, implementada en este mismo proyecto, y almacenamiento de su correspondiente salida en un array.

2. **Diferencia de *Sa* menos *Sb*:** Cálculo de la diferencia de Sa menos Sb mediante la llamada a la función del algoritmo de la diferencia, implementada en este mismo proyecto, y almacenamiento de su correspondiente salida en un array.

3. **Diferencia de *Sb* menos *Sa*:** Cálculo de la diferencia de Sb menos Sa mediante la llamada a la función del algoritmo de la diferencia, implementada en este mismo proyecto, y almacenamiento de su correspondiente salida en un array.

4. **Unión de las tres salidas anteriores:** Creación de un nuevo array para la salida definitiva del algoritmo de la unión y copia de los tres arrays calculados anteriormente en el mismo.

Capítulo 4

Implementación

En este capítulo se introducen los siguientes aspectos:

- Los mecanismos de abstracción utilizados en el proyecto para ocultar la complejidad del software; estructuras de datos, cabeceras y macros.
- La implementación de los funciones principales de los algoritmos de intersección, diferencia y unión, en base a las decisiones de diseño descritas en el capítulo 3.

4.1. Mecanismos de abstracción

En esta sección se describen las principales estructuras de datos empleadas para representar los tipos de datos complejos manejados por la librería Hitmap extendida. Además se describen las macros empleadas para sustituir funciones concretas o gestionar problemas relacionados con la alocación de memoria o la precisión de los dobles con los que trabajará la librería.

4.1.1. Estructuras de datos

- **Point**: Estructura de datos creada para modelar un punto para un espacio de $\mathbb{R}^2$. Este punto viene determinado por sus coordenadas reales, x e y, y un campo que indica si el punto es nulo o no (notnull).

  Surge por la necesidad de representar los puntos resultantes de las intersecciones entre los lados de los Shapes de entrada, ya que no pueden ser representados por Shapes puesto que su coordenadas son enteras y punto puede estar en cualquier posición del espacio para el espacio de $\mathbb{R}^2$.

```
typedef struct {
    double x;
    double y;
    int notnull; //0 es null, 1 no es null
} Point;
```

Listing 4.1: Estructura Point.

- **Segment**: Estructura de datos creada para modelar un segmento para un espacio de $\mathbb{R}^2$. Este segmento viene determinado por las coordenadas reales de sus extremos, x1-y1 y x2-y2 , y un campo que indica si el segmento es nulo o no (notnull).

  Surge por la necesidad de representar los segmentos que determinan los lados de los Shapes de entrada. A partir de estos segmentos se realizan las intersecciones entre los lados de los Shapes, mediante las ecuaciones de la rectas que contienen dichos segmentos y los rangos de $\mathbb{R}^2$ en los que están comprendidos. $\mathbb{R}^2$.

```
typedef struct {
    int type; // 0 base, 1 altura, 2 diagonal
    double x1;
    double y1;
    double x2;
    double y2;
    int notnull; //0 es null, 1 no es null
} Segment;
```

Listing 4.2: Estructura Segment.

- **PointLookUp**: Estructura da datos que extiende la estructura Point para almacenar la información generada durante la búsquedas de los triángulos de la partición del polígono de intersección. Esta estructura viene determinada por un punto del tipo Point, la dirección en la que ha sido alcanzado durante las búsquedas, un entero que indica si el punto resulta de una colisión al avanzar durante las búsquedas y en que dirección se ha producido dicha colisión y si es el resultado o no de la intersección con una diagonal.

  Surge de la necesidad de detectar cierta casuística para modelar el flujo de las búsquedas horizontal y verticales. En función de si el punto es generado como resultado de una colisión o no y en caso de serlo con una diagonal o no, supone diferencias a la hora de segmentos que deben almacenarse para la formación de triángulos de partición del polígono de intersección.

```
typedef struct {
    Point point;
    int direction; //1 (up), 2 (down), 3 (left) or 4 (right)
    int collision; //0 not result of a collision of segments, 1, 2,
        3, 4 result of collision and indicate direction of the
        collision
    int diagonal; //0 no result of intersection with diagonal.
        Otherwise, result of intersection with diagonal
} PointLookUp;
```

Listing 4.3: Estructura PointLookUp.

- **TriangleReal**: Estructura de datos que modela los triángulos que conforman la partición del polígono de intersección con vértices definidos para $\mathbb{R}^2$ resultante de las búsquedas horizontal y vertical. Esta estructura tiene cuatro campos que representan los tres vértices que determinan el triángulo, del tipo *PointLookUp*, teniendo que el vértice situado en el ángulo recto del triángulo se almacena dos veces. A mayores, se utiliza un campo para determinar si el triangulo es nulo o no.

  Surge de la necesidad de almacenar los triángulos determinados durante las búsquedas de triángulos de partición, los cuáles al manejar coordenadas reales no pueden ser modelados mediante shapes. Tras la fase de reajuste estos *triangelreal*s son reajustados para almacenarse mediante shapes.

```
typedef struct {
    PointLookUp A;
    PointLookUp B;
    PointLookUp Bortho;
    PointLookUp C;
    int notnull;
} TriangleReal;
```

Listing 4.4: Estructura TriangleReal.

4.2. Macros

Existen tres grupos de macros en la implementación de la propuesta del módulo de Hitmap:

- **Macros de gestión de alocación de memoria**

  A lo largo de la extensión de la librería se realizan múltiples reservas de memoria con el objetivo de alocar allí los distintos arrays de estructuras necesarias para operar con las entradas, almacenar los resultados intermedios de los algoritmos y dar la salida final en un único array de memoria.

  Con el objetivo de mejorar el rendimiento de los algoritmos y evitar la gestión de memoria de forma dinámica se han definido una serie de macros con valores enteros que representan el número de estructuras de memoria que se van a almacenar en los distintos arrays para los que se reserva memoria en los distintos algoritmos. Estas reservas de memoria, con el fin de evitar errores de punteros debido a valores no definidos se han realizado mediante la función *calloc* [33].

  Para el ejemplo del array de salida del algoritmo de intersección la reserva de memoria es de la forma:

```
HitSigExtShape * intersectionSaSb;
intersectionSaSb = calloc(ARRAY_TAM, sizeof(Point));
```

Mediante el uso de estas macros, en caso de que a través de la experimentación se observe que el tamaño de memoria reservada a lo largo del algoritmo es insuficiente, basta con sustituir el valor utilizado por las macros para la llamada a la función calloc.

```
#define ARRAY_TAM 16
#define ARRAY_TAM2 32
#define ARRAY_TAM3 64
#define ARRAY_TAM4 128
```

Listing 4.5: Macros de gestión de memoria alocada.

- **Macros de gestión de error de punto flotante**
  En el lenguaje de programación C, al realizar algunos cálculos con números decimales, por ejemplo con variables de tipo *double*, existe una cierta imprecisión debido a la representación binaria del número decimal, que puede que no sea exacta, y a la falta de coincidencia de tipos entre los números usados [34].

  Este tipo de errores se denominan errores de precisión en los cálculos de punto flotante [35]. Para el caso de los algoritmos desarrollados durante este proyecto, los errores de punto flotante han surgido en los cálculos de intersecciones entre los segmentos que determinan los lados de los shapes de entrada, generando bugs e incorrecciones en los cálculos realizados por los algoritmos. Estos errores se manifestaron al realizar la comprobación de si dos puntos tenían las mismas coordenadas: dos puntos teóricamente iguales en coordenadas resultaban ser diferentes por diferencias decimales de una precisión de 10^{-15}.

  Para solventar esto, se ha creado la macro *ERROR*, que determina que la precisión con la que va a trabar la librería es de 11 cifras decimales. En caso de que a través de la experimentación se demostrase de que esta precisión sigue generando incorreciones debido a problemas de cálculo de ounto flotante, dicho valor puede ser modificado.

```
#define ERROR 0.00000000001
```

Listing 4.6: Macros de gestión de error de punto flotante.

- **Macros que sustituyen funciones**
  El último grupo de macros implementadas son aquellas que sustituyen funciones. En concreto son dos, que realizan los cálculos del máximo y el mínimo de dos números.

  Estas macros surgen de la necesidad de realizar comparaciones entre números de forma frecuente, pudiendo ser estos de tipos distintos, ya sean dobles o enteros. Con estas macros el máximo y el mínimo se puede calcular sin realizar gestiones entre tipos.

```
#define MAX(a,b)    ((a) >= (b) ? (a) : (b))
#define MIN(a,b)    ((a) <= (b) ? (a) : (b))
```

Listing 4.7: Macros que calculan el máximo y el mínimo.

4.3. Funciones principales

- **intersection:** función que lleva a cabo el algoritmo intersección. Calcula el polígono de intersección entre dos shapes y devuelve la partición triangular de dicho polígono.

  - **Entradas:**
    Dos HitSigExtShape, Sa y Sb, que representan los polígonos de entradas (rectángulos o triángulos) para los cuáles se realiza la intersección.

  - **Salidas:**
    Un array de HitSigExtShape con los polígonos que conforman la partición del polígono de intersección.

  - **Funcionamiento:**
    Presenta dos fases principales: la primera es la construcción del polígono de intersección, donde se consiguen la lista de los puntos que representan los vértices que determinan el polígono de intersección, y la segunda fase, que es la partición del polígono de intersección, donde a partir de la lista de vértices se obtienen los triángulos que constituyen la partición y se devuelve como un array de HitSigExtShapes.

```
1  HitSigExtShape * intersection(HitSigExtShape Sa, HitSigExtShape Sb){
2
3      //Indica que el algoritmo es de interseccion
4    int difference = 0;
5
6      //Array de salida del algoritmo
7    HitSigExtShape * intersection;
8
9      //Se obtienen los vertices del poligono de interseccion
10   Point * vertex = intersection_polygon_Building(Sa, Sb, difference);
11
12     //Se obtiene la particion en triangulos del poligono
13     //de interseccion
14   intersection = ip_partition(vertex, Sa, Sb, difference);
15
16     //Se devuelve el array de salida con los Shapes
17     //que conforman la particion triangular del poligono
18     //de interseccion
19   return intersection;
20 }
21
```

Listing 4.8: Función principal del algoritmo de intersección.

- **difference:** función que lleva a cabo el algoritmo la diferencia. Calcula los polígonos resultantes de la diferencia entre dos shapes y devuelve la partición triangular de dichos polígonos.

  - **Entradas:**
    Dos HitSigExtShape, Sa y Sb, que representan los polígonos de entradas (rectángulos o triángulos) para los cuáles se realiza la diferencia.

  - **Salidas:**
    Un array de HitSigExtShape con los polígonos que conforman la partición de los polígonos resultantes de la diferencia entre los dos shapes.

  - **Funcionamiento:**
    Presenta tres fases principales: la primera es la construcción de los polígono resultantes de la diferencia, donde se consiguen la lista de los puntos que representan los vértices de dichos polígonos, la segunda fase, que es la partición de los polígonos resultantes de la diferencia, donde a partir de la lista de vértices se obtienen los triángulos que constituyen la partición y se devuelve como un array de HitSigExtShapes, y la tercera y última fase, donde los triángulos de la partición son separados del esqueleto de Sb para asegurar que la salida es disjunta entre sí y con Sb.

```
1
2  HitSigExtShape * difference(HitSigExtShape Sa, HitSigExtShape Sb){
3
4      //Indica que es el algoritmo de la diferencia
5    int differenceFlag = 1;
6
7      //Array de salida del algoritmo
8    HitSigExtShape * difference;
9
10     //Vertices que terminan los pol gonos resultantes de la diferencia
11     //entre Sa y Sb
12   Point * vertex = intersection_polygon_Building(Sa, Sb, differenceFlag);
13
14     //Se obtiene la particion en triangulos de los poligonos resultantes
15     //de la diferencia de Sa menos Sb
16   difference = ip_partition(vertex, Sa, Sb, differenceFlag);
17
18     //Se separan los Shapes resultantes del esquelo del Shape Sb
19     //para asegura que la salida y Sb son disjuntos
20   difference = separate_skeletons_from_Sb(difference, Sb);
21
22     //Se devuelve el array de salida con los Shapes que conforman
23     //la particion triangular de la diferencia del poligono de
24     //interseccion
25   return difference;
26 }
27
28
29
30
```

Listing 4.9: Función principal del algoritmo de la diferencia.

- **unionHitSig:** función que lleva a cabo el algoritmo de unión. Calcula los polígonos resultantes entre dos shapes y devuelve la partición triangular de dichos polígonos.

  - **Entradas:**
    Dos HitSigExtShape, Sa y Sb, que representan los polígonos de entradas (rectángulos o triángulos) para los cuáles se realiza la unión.

  - **Salidas:**
    Un array de HitSigExtShape con los polígonos que conforman la partición de los polígonos resultantes de la unión.

  - **Funcionamiento:**
    Presenta cuatro fases:

    1. En la primera se calcula la diferencia de Sa menos Sb con la función *difference* y se almacena en un array su correspondiente salida.
    2. En la segunda se calcula la diferencia de Sb menos Sa con la función *difference* y se almacena en un array su correspondiente salida.
    3. En la tercera se calcula la intersección entre Sa y Sb con la función *intersection* y se almacena en un array su correspondiente salida.
    4. En la cuarta y última fase se alamcenan las tres salidas anteriores en un único array que se corresponde con la salidad del algoritmo de unión.

```
1  HitSigExtShape * unionHitSig(HitSigExtShape Sa, HitSigExtShape Sb){
2
3      //Array donde se almacen los shapes que conforman
4      //la diferencia de Sa menos Sb
5    HitSigExtShape * differenceSaSb;
6
7      //Array donde se almacen los shapes que conforman
8      //la diferencia de Sb menos Sa
9    HitSigExtShape * differenceSbSa;
10
11     //Array donde se almacen los shapes que conforman
12     //la interseccion de Sa y Sb
13   HitSigExtShape * intersectionSaSb;
14
15     //Array de salida del algoritmo
16   HitSigExtShape * unionSaSb;
17
18     //Se calcula la diferencia de Sa menos Sb
19   differenceSaSb = difference(Sa, Sb);
20
21   //Se calcula la diferencia de Sb menos Sa
22   differenceSbSa = difference(Sb, Sa);
23
24   //Se calcula la interseccion de Sa y SB
25   intersectionSaSb = intersection(Sa, Sb);
26
27     //Reserva de memoria dela array de salida
28   unionSaSb = calloc(ARRAY_TAM4, sizeof(HitSigExtShape));
29
30   int z = 0;
```

```
31
32      //Se copian los Shapes de la diferencia
33      //de Sa menos Sb en el array de salida
34    for(int i=0; i<ARRAY_TAM2; i++){
35      if(!differenceSaSb[i].type==0 || !differenceSaSb[i].sig[0].begin == 0
36      || !differenceSaSb[i].sig[0].end == 0 || !differenceSaSb[i].sig[1].
        begin==0
37      || !differenceSaSb[i].sig[1].end ==0){
38        unionSaSb[z]= differenceSaSb[i];
39        z++;
40      }
41    }
42
43      //Se copian los Shapes de la diferencia
44      //de Sb menos Sa en el array de salida
45    for(int i=0; i<ARRAY_TAM2; i++){
46      if(!differenceSbSa[i].type==0 || !differenceSbSa[i].sig[0].begin == 0
47      || !differenceSbSa[i].sig[0].end == 0 || !differenceSbSa[i].sig[1].
        begin==0
48      || !differenceSbSa[i].sig[1].end ==0){
49        unionSaSb[z]= differenceSbSa[i];
50        z++;
51      }
52    }
53
54      //Se copian los Shapes de la interseccion
55      //de Sa y Sb en el array de salida
56    for(int i=0; i<ARRAY_TAM2; i++){
57      if(!intersectionSaSb[i].type==0 || !intersectionSaSb[i].sig[0].begin
        == 0
58      || !intersectionSaSb[i].sig[0].end == 0 || !intersectionSaSb[i].sig
        [1].begin==0
59      || !intersectionSaSb[i].sig[1].end ==0){
60        unionSaSb[z]= intersectionSaSb[i];
61        z++;
62      }
63    }
64
65      //Se devuelve el array de salida con
66      //los shapes que conforma la paritucion
67      //de la union de Sa y Sb
68    return unionSaSb;
69
70  }
71
72
73
```

Listing 4.10: Función principal del algoritmo de unión.

- **intersection_polygon_building:** función cuyo comportamiento varía en función de si es llamada por el algoritmo de la diferencia o por el algoritmo de intersección. Calcula los vértices que determinan el polígono de intersección o los polígonos resultantes de la diferencia de los shapes de entrada.

  - **Entradas:**
    Dos HitSigExtShape, Sa y Sb, que representan los polígonos de entradas (rectángu-

los o triángulos) para los cuáles se realizan la intersección o diferencia y un entero, que actúa como flag. Si toma el valor 0, el algoritmo que invoca a *intersection_polygon_building* es la intersección y si toma el valor 1 es la diferencia.

- **Salidas:**
  Un array de puntos que determinan el polígono de intersección o los polígonos de la diferencia, en función del algoritmo que invoca a *intersection_polygon_building*. Estos puntos son utilizados posteriormente en la función *ip_partition* durante las búsquedas horizontal y vertical.

- **Funcionamiento:**
  El funcionamiento depende de que algoritmo invoca esta función:
  - Para el algoritmo de intersección, se calculan las intersecciones entre los lados de los shapes de entrada, Sa y Sb, y se almacenan dichos puntos. Tras esto, se almacenan en el mismo array aquellos vértices de Sa que pertenecen a Sb y los vértices de Sb que pertenecen a Sa.
  - Para el algoritmo de la diferencia, se calculan las intersecciones entre los lados de los shapes de entrada, Sa y Sb, y se almacenan dichos puntos. Tras esto, se almacenan en el mismo array todos los vértices de Sa y aquellos vértices de Sb que pertenezcan a Sa.

```
1
2  Point * intersection_polygon_Building(HitSigExtShape Sa, HitSigExtShape
       Sb, int difference){
3
4      //Array de salida que almacenara los vertices del
5      //poligono de interseccion o los vertices que determinan
6      //los poligonos de la diferencia
7    Point * puntosPoligonoInterseccion;
8
9      //Arrays que almacenan los vertices de Sa y Sb
10   Point * vertexSa;
11   Point * vertexSb;
12
13     //Se reserva memoria para el array de vertices
14   puntosPoligonoInterseccion = calloc(ARRAY_TAM, sizeof(Point));
15
16     //Punto donde se almacenara las intersecciones
17     //calculadas en cada iteracion
18   Point auxpoint;
19   int z = 0;
20
21     //Arrays que almacenan los segmentos de Sa y Sb
22   Segment * segmentsSa;
23   Segment * segmentsSb;
24
25     //Shape auxiliar para intercambiar Sa con Sb
26   HitSigExtShape Sc;
27
28     //En caso de que Sa sea un rectangulo y Sb
29     //un triangulo, estos se intercambian
30   if(Sa.type == 0 && Sb.type == 1){
31     Sc = Sa;
```

```
32      Sa = Sb;
33      Sb = Sc;
34    }
35
36      //Obtenemos los segmentos que determinan los
37      //lados de los shapes de entrada
38    segmentsSa = get_segments_from_Shape(Sa);
39    segmentsSb = get_segments_from_Shape(Sb);
40
41      //Obtenemos los vertices de los shapes Sa y Sb
42    vertexSa = get_points_from_Shape(Sa);
43    vertexSb = get_points_from_Shape(Sb);
44
45      //Interseccion entre 2 rectangulos
46    if(Sa.type == 0 && Sb.type == 0){
47
48          //Intersecamos todos los lados no paralelos entre si
49      for(int i=0; i<4; i++){
50        for(int j=0; j<4; j++){
51
52          if(segmentsSa[i].type == 0 && segmentsSb[j].type == 1){ //
      intersection base Sa height Sb
53
54            auxpoint = orthogonal_intersection(segmentsSa[i], segmentsSb[j
      ]);
55
56            //Si hay interseccion se almacena
57            if(auxpoint.notnull == 1){
58
59              z++;
60              puntosPoligonoInterseccion[z-1] = auxpoint;
61
62            }
63
64          }else if(segmentsSb[j].type == 0 && segmentsSa[i].type == 1){ //
      intersection base Sb height Sa
65
66            auxpoint = orthogonal_intersection(segmentsSb[j], segmentsSa[i
      ]);
67
68                      //Si hay interseccion se almacena
69            if(auxpoint.notnull == 1){
70
71              z++;
72              puntosPoligonoInterseccion[z-1] = auxpoint;
73
74            }
75
76          }
77
78        }
79
80      }
81
82      //interseccion entre un triangulo y un rectangulo
83    }else if(Sa.type == 1 && Sb.type == 0){
84
85          //Intersecamos todos los lados no paralelos entre si
86      for(int i=0; i<3; i++){
```

```
87        for(int j=0; j<4; j++){
88
89          if(segmentsSa[i].type == 0 && segmentsSb[j].type == 1){ //
      intersection base Sa height Sb
90
91            auxpoint = orthogonal_intersection(segmentsSa[i], segmentsSb[j
      ]);
92
93            //Si hay interseccion se almacena
94            if(auxpoint.notnull == 1){ //hay inteseccion
95
96              z++;
97              puntosPoligonoInterseccion[z-1] = auxpoint;
98
99            }
100
101         }else if(segmentsSb[j].type == 0 && segmentsSa[i].type == 1){ //
      intersection base Sb height Sa
102
103           auxpoint = orthogonal_intersection(segmentsSb[j], segmentsSa[i
      ]);
104
105           //Si hay interseccion se almacena
106           if(auxpoint.notnull == 1){
107
108             z++;
109             puntosPoligonoInterseccion[z-1] = auxpoint;
110
111           }
112
113         }else if(segmentsSa[i].type == 2){ //intersection diagonal Sa,
      height or base Sb
114
115           auxpoint = diagonal_intersection(segmentsSa[i], segmentsSb[j]);
116
117           //Si hay interseccion se almacena
118           if(auxpoint.notnull == 1){
119
120             z++;
121             puntosPoligonoInterseccion[z-1] = auxpoint;
122
123           }
124
125         }
126
127       }
128
129     }
130
131     //interseccion entre dos triangulos
132   }else{
133
134         //Intersecamos todos los lados no paralelos entre si
135     for(int i=0; i<3; i++){
136       for(int j=0; j<3; j++){
137
138         if(segmentsSa[i].type == 0 && segmentsSb[j].type == 1){ //
      intersection base Sa height Sb
139
```

```
140             auxpoint = orthogonal_intersection(segmentsSa[i], segmentsSb[j
    ]);
141
142             //Si hay interseccion se almacena
143             if(auxpoint.notnull == 1){
144
145               z++;
146               puntosPoligonoInterseccion[z-1] = auxpoint;
147
148             }
149
150         }else if(segmentsSb[j].type == 0 && segmentsSa[i].type == 1){ //
    intersection base Sb height Sa
151
152             auxpoint = orthogonal_intersection(segmentsSb[j], segmentsSa[i
    ]);
153
154                       //Si hay interseccion se almacena
155             if(auxpoint.notnull == 1){
156
157               z++;
158               puntosPoligonoInterseccion[z-1] = auxpoint;
159
160             }
161
162         }else if(segmentsSa[i].type == 2){ //intersection diagonal Sa
    segment of Sb
163
164             auxpoint = diagonal_intersection(segmentsSa[i], segmentsSb[j]);
165
166             //Si hay interseccion se almacena
167             if(auxpoint.notnull == 1){
168
169               z++;
170               puntosPoligonoInterseccion[z-1] = auxpoint;
171
172             }
173
174         }else if((segmentsSb[j].type == 2)&&(segmentsSa[i].type != 2)){
    //intersection diagonal Sb segment of Sa
175
176             auxpoint = diagonal_intersection(segmentsSb[j], segmentsSa[i]);
177
178             //Si hay interseccion se almacena
179             if(auxpoint.notnull == 1){
180
181               z++;
182               puntosPoligonoInterseccion[z-1] = auxpoint;
183
184             }
185
186         }
187
188       }
189
190     }
191
192   }
193
```

```
194
195     //Se a ade al array de vertices del poligono de interseccion
196     //Aquellos vertices de Sa que pertenecen a Sb
197   for(int i = 0; (i < 4)&&(vertexSa[i].notnull==1); i++){
198
199     if(point_belongs_to_Shape(vertexSa[i],Sb)){
200
201       z++;
202       puntosPoligonoInterseccion[z-1] = vertexSa[i];
203
204         //Si el algoritmo es de la diferencia se a aden
205         //todos los vertices de Sa, no solo los que pertenecen
206         //a Sb
207     }else if(difference){
208
209       z++;
210       puntosPoligonoInterseccion[z-1] = vertexSa[i];
211
212     }
213
214   }
215
216     //Se a ade al array de vertices del poligono de interseccion
217     //Aquellos vertices de Sb que pertenecen a Sa
218   for(int i = 0; (i < 4)&&(vertexSb[i].notnull==1); i++){
219
220     if(point_belongs_to_Shape(vertexSb[i],Sa)){
221
222       z++;
223       puntosPoligonoInterseccion[z-1] = vertexSb[i];
224
225     }
226
227   }
228
229     //Se reordenan los puntos en el array
230     //Primero los puntos con menor coordenada y
231     //y con menor coordenada x
232   puntosPoligonoInterseccion = order_points(puntosPoligonoInterseccion);
233
234     //Se devuelve el array de salida con los v rtices del
235     //poligono de interseccion o los vertices de los poligonos
236     //de la diferencia
237   return puntosPoligonoInterseccion;
238
239 }
240
241
```

Listing 4.11: Función principal de construcción del polígono de intersección.

- **ip_partition:** función cuyo comportamiento varía en función de si es llamada por el algoritmo de la diferencia o por el algoritmo de intersección. Calcula la partición en triángulos del polígono de intersección o de los polígonos resultantes de la diferencia de los shapes de entrada.

  - **Entradas:**
    Dos HitSigExtShape, Sa y Sb, que representan los polígonos de entradas (rectángu-

los o triángulos) para los cuáles se realizan la intersección o diferencia y un entero, que actúa como flag. Si toma el valor 0, el algoritmo que invoca a *ip_partition* es la intersección y si toma el valor 1 es la diferencia.

- **Salidas:**
  Un array de HitSigExtShapes que representa la partición del polígono de intersección o de los polígonos resultantes de la diferencia entre Sa y Sb.

- **Funcionamiento:**
  El funcionamiento depende de que algoritmo invoca esta función:
  - Para el algoritmo de intersección, se realizan las búsquedas horizontal y vertical en aquellos puntos interiores a los shapes Sa y Sb. Tras localizar aquellos puntos que determinan los vértices de los triángulos de la partición de la intersección, se ordenan, se obtienen aquellos puntos y segmentos aislados que no determinan un triángulo y con los vértices restantes se constituyen los triángulos como shapes. Por último, se reajustan dichos shapes para asegurar que sean disjuntos.
  - Para el algoritmo de la diferencia, se realizan las búsquedas horizontal y vertical, dos veces cada una, en aquellos puntos interiores al shape Sa que no pertenezcan al interior de Sb. Tras localizar aquellos puntos que determinan los vértices de los triángulos de la partición de los polígonos resultantes, se ordenan, se obtienen aquellos puntos y segmentos aislados que no determinan un triángulo y con los vértices restantes se constituyen los triángulos como shapes. Por último, se reajustan dichos shapes para asegurar que sean disjuntos.

```
1
2  HitSigExtShape * ip_partition(Point vertex[], HitSigExtShape Sa,
       HitSigExtShape Sb, int difference){
3
4      //Array de salida que almacenara los shapes que conforman
5      //la particion triangular de la interseccion o la diferencia
6      //de Sa y Sb
7    HitSigExtShape * intersection;
8    intersection = calloc(ARRAY_TAM2, sizeof(HitSigExtShape));
9
10     //Array que almacenara los puntos generados durante las
11     //busquedas y que determinan los triangulos de la particion
12   PointLookUp * visitedPoints;
13   visitedPoints = calloc(ARRAY_TAM4, sizeof(PointLookUp));
14
15     //Array que almacenara los puntos generados durante las
16     //busquedas aplicadas a mayores en la diferencia
17   Point * visitedPointsDifference;
18   visitedPointsDifference = calloc(ARRAY_TAM4, sizeof(PointLookUp));
19
20     //Array que almacena los segementos visitados durante las
21     //busquedas horizontal y vertical
22   Segment * visitedSegments;
23   visitedSegments = calloc(ARRAY_TAM2, sizeof(Segment));
24
25     //Busqueda horizontal
```

```
26      visitedPoints = horizontal_lookup(vertex, Sa, Sb, visitedPoints,
        visitedSegments, difference);
27
28      //Si el algoritmo es de diferencia se vuelve a aplicar
29      //la busqueda
30      if(difference){
31
32        for(int i = 0; i< ARRAY_TAM4 && visitedPoints[i].point.notnull; i
        ++){
33
34          visitedPointsDifference[i] = visitedPoints[i].point;
35
36        }
37
38        visitedPoints = horizontal_lookup(visitedPointsDifference, Sa, Sb,
        visitedPoints, visitedSegments, difference);
39      }
40
41      //Busqueda vertical
42      visitedPoints = vertical_lookup(vertex, Sa, Sb, visitedPoints,
        visitedSegments, difference);
43
44      //Si el algoritmo es de diferencia se vuelve a aplicar
45      //la busqueda
46      if(difference){
47
48        for(int i = 0; i< ARRAY_TAM4 && visitedPoints[i].point.notnull; i
        ++){
49
50          visitedPointsDifference[i] = visitedPoints[i].point;
51
52        }
53
54        visitedPoints = vertical_lookup(visitedPointsDifference, Sa, Sb,
        visitedPoints, visitedSegments, difference);
55      }
56
57      //Se ordenan los puntos del array
58      //Primero los puntos con menor coordenada Y
59      //y con menor coordenada X
60      visitedPoints = order_Look_Up_points(visitedPoints);
61
62      //Se buscan los puntos y segmentos aislados entre los puntos
        obtenidos
63      //que no podran determinar un triangulo de la particion
64      intersection = isolated_vertices_and_segments(intersection, vertex,
        visitedPoints, Sa, Sb);
65
66      //Se obtienen los triangulos que determinan la particion
67      //del poligono de interseccion o de los poligonos de la
68      //diferencia
69      intersection = determine_triangles(intersection, visitedPoints);
70
71      //Se separan los esqueletos de los triangulos para asegurar
72      //que la particion es disjunta
73      intersection = separate_skeletons(intersection);
74
75      //Se devuelve el array de salida con los poligonos
76      //de la particion
```

```
77    return intersection;
78
79 }
80
81
```

Listing 4.12: Función principal de la partición del polígono de intersección.

4.4. Funciones secundarias

- **point_is_in_the_array:** Comprueba si el punto introducido por parámetro está presente en el array introducido como entrada. El valor 1 de la salida indica que el punto está presente, por lo contrario toma el valor 0.

```
int point_is_in_the_array(Point point, Point * pointList);
```

- **m_and_t_of_segment:** Calcula la pendiente y el punto de corte con el eje de abscisas de la recta determinada por dos puntos introducidos por parámetro.

```
void m_and_t_of_segment(Point A, Point C, double *m, double *t);
```

- **diagonal:** Calcula la coordenada Y de un punto con coordenada X, introducida por parámetro, que pertenece a la recta determinada por los dos puntos introducidos como parámetros.

```
double diagonal(Point A, Point C, double x);
```

- **same_coordinates:** Comprueba si los puntos introducidos por parámetro tienen las mismas coordenadas. El valor 1 de la salida indica que ambos puntos tienen las mismas coordenadas, por lo contrario toma el valor 0.

```
int same_coordinates(Point point1, Point point2);
```

- **shape_to_point:** Construye el punto que representa el shape introducido por parámetro.

```
Point shape_to_point(HitSigExtShape S);
```

- **get_type_from_triangle:** Comprueba que tipo de triángulo es el triángulo introducido por parámetro. Puede ser de tipo 1, 2, 3 o 4.

```
int get_type_from_triangle(HitSigExtShape S)
```

- **get_type_from_triangle_real:** Comprueba que tipo de triángulo es el triángulo introducido por parámetro. Puede ser de tipo 1, 2, 3 o 4.

```
int get_type_from_triangle_real(TriangleReal triangle);
```

- **get_points_from_Shape:** Obtiene un array de puntos con los vértices del shape introducido por parámetro.

```
Point * get_points_from_Shape(HitSigExtShape S);
```

- **get_segments_from_Shape:** Obtiene un array de segmentos con los lados del shape introducido por parámetro.

```
Segment * get_segments_from_Shape(HitSigExtShape S);
```

- **get_diagonal:** Obtiene la diagonal del array de segmentos pasado por parámetro, que previamente ha sido obtenido con la función *get_segments_from_Shape*.

```
Segment get_diagonal(Segment segments[]);
```

- **point_belongs_to_segment:** Comprueba si un punto pertenece al segmento introducido por parámetro. Devuelve el valor 1 si dicho punto pertenece y 0 en el caso contrario.

```
int point_belongs_to_segment(Point point, Segment segment);
```

- **point_belongs_to_Shape:** Comprueba si un punto pertenece al shape introducido por parámetro. Devuelve el valor 1 si dicho punto pertenece y 0 en el caso contrario.

```
int point_belongs_to_Shape(Point point, HitSigExtShape S);
```

- **order_points:** Ordena los puntos introducidos en el array de entrada de forma que los puntos con menor coordenada Y y menor coordenada X vayan primero.

```
Point * order_points(Point points[]);
```

- **orthogonal_intersection:** Calcula la intersección entre dos segmentos perpendiculares introducidos por parámetro.

```
Point orthogonal_intersection(Segment s1, Segment s2);
```

- **diagonal_intersection:** Calcula la intersección entre dos segmentos introducidos por parámetro, donde al menos el primero es diagonal.

```
Point diagonal_intersection(Segment s1, Segment s2);
```

- **min:** Dados cuatro valores enteros introducidos como entrada devuelve el mínimo de los mismos.

```
double min(int a, int b, int c, int d);
```

- **max:** Dados cuatro valores enteros introducidos como entrada devuelve el máximo de los mismos.

```
double max(int a, int b, int c, int d);
```

- **almost_equal:** Comprueba si dos valores de tipo *double* tienen una diferencia menor a la indicada por la macro ERROR. Devuelve el valor 1 si la diferencia es menor que ERROR y 0 en el caso contrario.

```
int almost_equal(double a, double b);
```

- **order_Look_Up_points:** Ordena los puntos extendidos introducidos en el array de entrada de forma que los puntos con menor coordenada Y y menor coordenada X vayan primero.

```
PointLookUp * order_Look_Up_points(PointLookUp points[]);
```

- **collision:** Dado un punto extendido generado al colisionar con un segmento previamente visitado en el avance de alguna de las búsquedas, obtiene el punto extendido de origen del avance que junta al punto de entrada determina el segmento a almacenar.

```
PointLookUp collision(PointLookUp collidedPoint, int direction,
    PointLookUp visitedPoints[]);
```

- **point_belongs_to_skeleton:** Comprueba si el punto introducido por parámetro pertenece al perímetro del shape introducido por parámetro. Devuelve el valor 1 si el punto pertenece y 0 en el caso contrario.

```
int point_belongs_to_skeleton(Point point, HitSigExtShape S);
```

- **segment_is_on_border:** Comprueba si el segmento introducido por parámetro pertenece al perímetro del shape introducido por parámetro. Devuelve el valor 1 si el segmento pertenece y 0 en el caso contrario.

```
int segment_is_on_border(Point point1, Point point2, HitSigExtShape Sa,
    HitSigExtShape Sb);
```

- **point_already_visited:** Comprueba si el punto introducido por parámetro ya ha sido visitado durante las búsquedas horizontal y vértical. Devuelve el valor 1 si el punto ya ha sido visitado y 0 en el caso contrario.

```
int point_already_visited(Point point, int direction, PointLookUp
    visitedPoints[]);
```

- **diagonal_loop:** Comprueba si existe un punto con coordenadas enteras (un índice de memoria) entre los dos puntos introducidos por parámetro. Devuelve el valor 1 si existe dicho punto y 0 en el caso contrario.

```
int diagonal_loop(Point point, Point newPoint, int direction);
```

- **try_to_advance:** Dados un punto de partida, una dirección, los shapes de entrada de los algoritmos de intersección, diferencia o unión, los lados de los mismos y un array con los segementos ya visitados calcula el siguiente punto alcanzado durante el avance en la fase de búsquedas.

```
PointLookUp try_to_advance(Point point, int direction, HitSigExtShape Sa,
    HitSigExtShape Sb, Segment segmentsSa[], Segment segmentsSb[],
    PointLookUp visitedPoints[], Segment visitedSegments[], int
    difference);
```

- **horizontal_lookup:** Dados un punto de partida, una dirección, los shapes de entrada de los algoritmos de intersección, diferencia o unión, los lados de los mismos, un array con los segmentos ya visitados y otro con los puntos ya visitados obtiene parte de los vértices que determinan los triángulos de la partición.

```
PointLookUp * horizontal_lookup(Point vertex[], HitSigExtShape Sa,
    HitSigExtShape Sb, PointLookUp visitedPoints[], Segment
    visitedSegments[], int difference);
```

- **vertical_lookup:** Dados un punto de partida, una dirección, los shapes de entrada de los algoritmos de intersección, diferencia o unión, los lados de los mismos, un array con los segmentos ya visitados y otro con los puntos ya visitados obtiene parte de los vértices que determinan los triángulos de la partición.

```
PointLookUp * vertical_lookup(Point vertex[], HitSigExtShape Sa,
    HitSigExtShape Sb, PointLookUp visitedPoints[], Segment
    visitedSegments[], int difference);
```

- **point_to_Shape:** Construye un shape que representa el punto introducido por parámetro.

```
HitSigExtShape point_to_Shape(Point point);
```

- **segment_to_Shape:** Construye un shape que representa el segmento introducido por parámetro.

```
HitSigExtShape segment_to_Shape(Point point1, Point point2);
```

- **isolated_vertices_and_segments:** Dado el array de puntos alcanzados durante las búsquedas horizontal y vertical, detecta aquellos puntos y segmentos aislados que no forman parte de un triángulo de la partición y se almacenan en el array de salida representados mediante shapes.

```
HitSigExtShape * isolated_vertices_and_segments(HitSigExtShape
    intersection[], Point vertex[], PointLookUp visitedPoints[],
    HitSigExtShape Sa, HitSigExtShape Sb);
```

- **find_point:** Dado el array de puntos visitados durante las búsquedas y un punto introducido por parámetro determina si dicho punto se encuentra entre los puntos visitados.

```
PointLookUp find_point(PointLookUp point, PointLookUp visitedPoints[],
    int size);
```

- **orthogonal_point:** Dado el array de puntos visitados durante las búsquedas y un punto extendido, devuelve un punto de dicho array con las mismas coordenadas y dirección ortogonal que determina junto al primero el ańgulo rectángulo de uno de los triángulos de la partición.

```
PointLookUp orthogonal_point(PointLookUp point, PointLookUp visitedPoints
    [], int size);
```

- **two_segments_to_triangle_real:** Dados los cuatro puntos que determinan los dos segmentos que representan la base y altura de un triángulo, se devuelve el triángulo de coordenadas reales determinado por estos.

```
TriangleReal two_segments_to_triangle_real(PointLookUp point1,
    PointLookUp point2, PointLookUp point3, PointLookUp point4);
```

- **ceil_or_add:** Redondea una coordenada de un shape al entero superior.

```
int ceil_or_add(double x);
```

- **floor_or_subs:** Redondea una coordenada de un shape al entero inferior.

```
int floor_or_subs(double x);
```

- **outsiders:** Dados un triángulo de coordenadas reales y el shape resultante del reajuste del primero a coordenadas enteras, obtiene un array con los puntos que pertenecían al triángulo inicial pero que se pierden al reajustar.

```
HitSigExtShape * outsiders(TriangleReal triangleOriginal, HitSigExtShape
    triangleModified);
```

- **adjust_triangle:** Dados los 3 vértices (uno duplicado pero con direcciones distintas) que conforman un triángulo con coordenadas reales, reajusta dicho triángulo a coordenadas enteras y es devuelto mediante la estructura shape.

```
HitSigExtShape adjust_triangle(PointLookUp point1, PointLookUp point2,
    PointLookUp point3, PointLookUp point4);
```

- **point_belongs_to_any_Segment:** Dado un array de segmentos y un punto, determina si dicho punto pertenece a alguno de los segmentos. En caso afirmativo se devuelve el valor 1 y en caso contrario el valor 0.

```
int point_belongs_to_any_Segment(Point point, Segment segments[]);
```

- **two_points_to_segment:** Dados dos puntos devuelve el segmento que estos determinan.

```
Segment two_points_to_segment(Point point1, Point point2);
```

- **find_and_null:** Dado el array de puntos visitados durante las búsquedas, esta función localiza los trs puntos introducidos por parámetro en el array y los fija como nulos.

```
PointLookUp * find_and_null(PointLookUp point1, PointLookUp point2,
    PointLookUp point3, PointLookUp * visitedPoints, int size);
```

- **triangle_overlaps_triangle:** Dados dos triángulos introducidos por parámetro comprueba si presentan puntos comunes. En caso afirmativo se devuelve el valor 1 y en caso contrario el valor 0.

```
int triangle_overlaps_triangle(TriangleReal triangle1, TriangleReal
    triangle2);
```

- **triangle_overlaps_any_triangle:** Dados un triángulo y un array de triángulos introducidos por parámetros comprueba si el primero se solapa con alguno de los triángulos presentes en el array. En caso afirmativo se devuelve el valor 1 y en caso contrario el valor 0.

```
int triangle_overlaps_any_triangle(TriangleReal triangle, TriangleReal
    triangles[]);
```

- **determine_triangles:** Dado el array de puntos obtenidos durante las búsquedas, construye los triángulos de la partición del polígono de la intersección o de los polígonos resultantes de la diferencia y los almacena en el array de salida de los algoritmos.

```
HitSigExtShape * determine_triangles(HitSigExtShape intersection[],
    PointLookUp visitedPoints[]);
```

- **join_rectangles:** Dado el array con los shapes de salida de los algoritmos de intersección, diferencia o unión, realiza la fusión de aquellos shapes que son rectángulos contiguos y que van a generar como resultado un nuevo rectángulo más grande.

```
int join_rectangles(HitSigExtShape * intersection, int * valid, int i,
    int j, HitSigExtShape Sb, int check_cross);
```

- **outsider_is_redundant:** Comprueba si un outsider introducido por parámetro pertenece a otro shape de entrada. En caso afirmativo se devuelve el valor 1 y 0 en caso contrario.

```
int outsider_is_redundant(HitSigExtShape outsider, HitSigExtShape figure)
    ;
```

- **share_interval:** Dados dos HitSigExt comprueba si los intervalos que representan ambas estructuras comparten intervalos. En caso afirmativo se devuelve el valor 1 y 0 en caso contrario.

```
int share_interval(HitSigExt Sa, HitSigExt Sb);
```

- **separate_skeletons_rect_rect:** Dados dos rectángulos introducidos por parámetros, se realiza el reajuste de las coordenadas del primero para asegurar que estos sean disjuntos.

```
int separate_skeletons_rect_rect(HitSigExtShape intersection[], int valid
    [], int i, int j, int * z);
```

- **separate_skeletons_rect_triangle:** Dados un rectángulo y un triángulo introducidos por parámetros, se realiza el reajuste de las coordenadas del primero para asegurar que estos sean disjuntos.

```
int separate_skeletons_rect_triangle(HitSigExtShape intersection[], int
    valid[], int i, int j, int * z);
```

- **modify_diagonal:** Dado un triángulo introducido por parámetro, modifica su lado diagonal reajustando su vértice A o C o ambos en función de los flags de entrada.

```
void modify_diagonal(HitSigExtShape *triangle, int modA, int modC);
```

- **outsiders_separate_skeletons:** Dados un triangulo de coordenadas reales y el shape resultante del reajuste del primero a coordenadas enteras, obtiene un array con los puntos que pertenecían al triángulo inicial pero que se pierden al reajustar.

```
HitSigExtShape * outsiders_separate_skeletons(TriangleReal
    triangleOriginal, HitSigExtShape triangleModified, int admitDiag, int
     x, int y);
```

- **separate_skeletons_triangle_triangle:** Dados dos triángulos introducidos por parámetros, se realiza el reajuste de las coordenadas del primero para asegurar que estos sean disjuntos.

```
int separate_skeletons_triangle_triangle(HitSigExtShape intersection[],
    int valid[], int i, int j, int * z);
```

- **separate_skeletons:** Dados dos shapes introducidos por parámetro se realiza el reajuste de las coordenadas de los mismos para asegurar que estos son disjuntos y no comparten puntos en su perímetro.

```
HitSigExtShape * separate_skeletons(HitSigExtShape intersection[],
    HitSigExtShape Sb);
```

- **belonging_condition_difference:** Comprueba si un punto alcanzado durante las búsquedas desde otro punto inicial, ambos introducidos como entrada, pertenece a la diferencia de Sa menos Sb.

```
int belonging_condition_difference(Point initialPoint, Point reachedPoint
    , HitSigExtShape Sa, HitSigExtShape Sb);
```

- **belonging_condition:** Comprueba si un punto alcanzado durante las búsquedas desde otro punto inicial, ambos introducidos como entrada, pertenece a la intersección entre Sa y Sb

```
int belonging_condition(Point initialPoint, Point reachedPoint,
    HitSigExtShape Sa, HitSigExtShape Sb, int difference);
```

- **separate_skeletons_triangle_rect:** Dados un triángulo y un rectángulo introducidos por parámetros, se realiza el reajuste de las coordenadas del primero para asegurar que estos sean disjuntos.

```
int separate_skeletons_triangle_rect(HitSigExtShape intersection[], int
    valid[], int i, int j, int * z);
```

- **separate_skeletons_from_Sb:** Dado un array de shapes y otro shape Sb, introducidos por parámetros, se realiza el reajuste de las coordenadas de todos los shapes del array para asegurar que estos sean disjuntos con respecto a Sb.

```
HitSigExtShape * separate_skeletons_from_Sb(HitSigExtShape intersection
    [], HitSigExtShape Sb);
```

- **readjust_triangle_separate:** Realiza el reajuste de las coordenadas de un triángulo introducido por parámetro, modificando los vértices A o B o C del mismo o cualquier combinación de los mismos, en función de los flags de entrada.

```
HitSigExtShape readjust_triangle_separate(HitSigExtShape triangle, int
    readjustA, int readjustB, int readjustC, double m, double t);
```

Capítulo 5

Pruebas

En este capítulo se introducen los siguientes aspectos:

- Se detallan los casos de prueba más significativos planteados para cada uno de los algoritmos implementados (intersección, diferencia y unión), así como sus salidas esperadas y las salidas obtenidas.

Las figuras ilustradas durante este capítulo siguen la nomenclatura indicada en la sección 3.2.

5.1. Pruebas

En esta sección se muestran los casos de prueba más significativos empleados en los algoritmos de intersección, diferencia y unión para comprobar la validez de los mismos. Estos casos de prueba se dividen en dos tipos; los casos base y los casos límite.

Para aquellos casos en los que se obtenga una salida incorrecta se localiza la causa de dicho error, se realizan las modificaciones pertinentes y se documentan con el objetivo de obtener un software funcional y usable por el grupo de investigación Trasgo.

5.1.1. Casos base

Los casos base de un conjuntos de pruebas de validación son aquellos que buscan comprobar la corrección del funcionamiento de un software para un conjunto de entradas estándar, es decir, aquellas entradas que son esperables/deseables y que muestran si los cálculos realizados por los algoritmos son correctos para la mayoría de situaciones.

- **Caso de prueba 1**

  Dos triángulos como entrada que determinan un polígono de intersección cuyos vértices distan entre sí una distancia superior a una unidad en el plano de coordenadas para 2 dimensiones y entre los que está comprendido al menos un punto de coordenadas enteras.

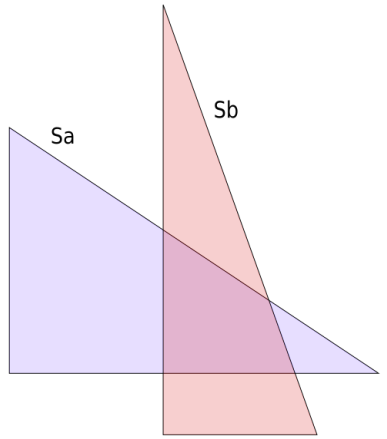

Figura 5.1: Caso de prueba 1.

- **Shapes de entrada**

  $Sa = \{2, 1, \{0, 12, 1\}, \{0, 8, 1\}\}$
  $Sb = \{2, 1, \{5, 10, 1\}, \{-2, 12, 1\}\}$

- **Salida del caso de prueba 1 para el algoritmo de intersección**

```
Caso 1 - intersection

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {5, 10, 1}, {-2, 12, 1}}

Shapes esperados:
shapes[0] = {2, 0, {5, 8, 1}, {0, 2, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {0, 0, 1}}
shapes[2] = {2, 1, {5, 7, 1}, {3, 4, 1}}
shapes[3] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Shapes obtenidos:
shapes[0] = {2, 0, {5, 8, 1}, {0, 2, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {0, 0, 1}}
shapes[2] = {2, 1, {5, 7, 1}, {3, 4, 1}}
shapes[3] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Resultado: Correcto
```

Listing 5.1: Salidas esperadas y obtenidas por la intersección para el caso 1.

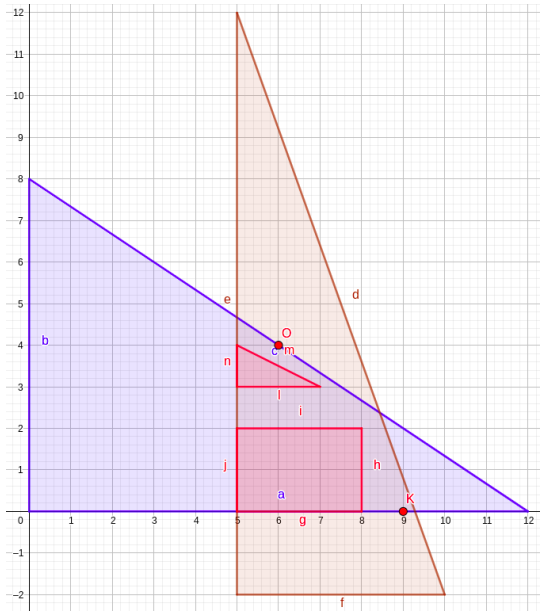

Figura 5.2: Salida del caso de prueba 1 para el algoritmo de intersección.

- **Salida del caso de prueba 1 para el algoritmo de la diferencia**

```
Caso 1 - difference

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {5, 10, 1}, {-2, 12, 1}}

Shapes esperados:
shapes[0] = {2, 0, {0, 4, 1}, {0, 4, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {1, 1, 1}}
shapes[2] = {2, 1, {10, 12, 1}, {0, 1, 1}}
shapes[3] = {2, 1, {9, 9, 1}, {2, 2, 1}}
shapes[4] = {2, 1, {0, 4, 1}, {5, 8, 1}}
shapes[5] = {2, 1, {3, 3, 1}, {6, 6, 1}}

Shapes obtenidos:
shapes[0] = {2, 0, {0, 4, 1}, {0, 4, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {1, 1, 1}}
shapes[2] = {2, 1, {10, 12, 1}, {0, 1, 1}}
shapes[3] = {2, 1, {9, 9, 1}, {2, 2, 1}}
shapes[4] = {2, 1, {0, 4, 1}, {5, 8, 1}}
shapes[5] = {2, 1, {3, 3, 1}, {6, 6, 1}}

Resultado: Correcto
```

Listing 5.2: Salidas esperadas y obtenidas por la diferencia para el caso 1.

- **Salida del caso de prueba 1 para el algoritmo de unión**

```
Caso 1 - unionHitSig
```

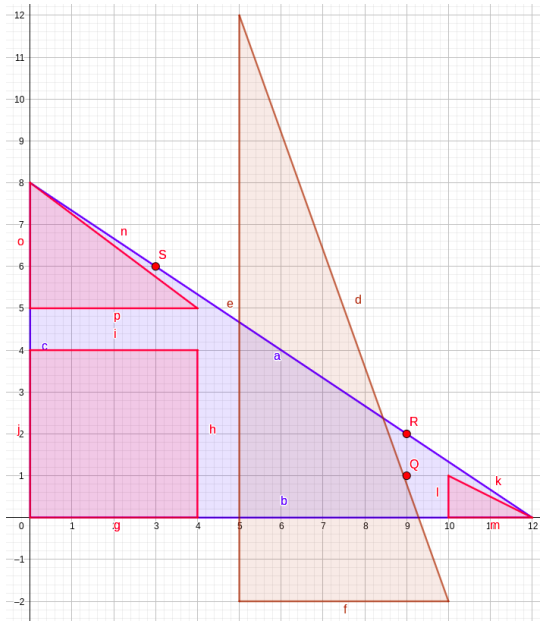

Figura 5.3: Salida del caso de prueba 1 para el algoritmo de la diferencia.

```
Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {5, 10, 1}, {-2, 12, 1}}

Shapes esperados:
shapes[0] = {2, 0, {0, 4, 1}, {0, 4, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {1, 2, 1}}
shapes[2] = {2, 1, {10, 12, 1}, {0, 1, 1}}
shapes[3] = {2, 1, {0, 4, 1}, {5, 8, 1}}
shapes[4] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[5] = {2, 0, {5, 9, 1}, {-2, -1, 1}}
shapes[6] = {2, 1, {10, 10, 1}, {-2, -2, 1}}
shapes[7] = {2, 1, {7, 7, 1}, {4, 4, 1}}
shapes[8] = {2, 1, {8, 8, 1}, {3, 3, 1}}
shapes[9] = {2, 1, {5, 7, 1}, {5, 12, 1}}
shapes[10] = {2, 1, {6, 6, 1}, {9, 9, 1}}
shapes[11] = {2, 1, {7, 7, 1}, {6, 6, 1}}
shapes[12] = {2, 0, {5, 8, 1}, {0, 2, 1}}
shapes[13] = {2, 1, {9, 9, 1}, {0, 0, 1}}
shapes[14] = {2, 1, {5, 7, 1}, {3, 4, 1}}
shapes[15] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Shapes obtenidos:
shapes[0] = {2, 0, {0, 4, 1}, {0, 4, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {1, 1, 1}}
shapes[2] = {2, 1, {10, 12, 1}, {0, 1, 1}}
shapes[3] = {2, 1, {9, 9, 1}, {2, 2, 1}}
shapes[4] = {2, 1, {0, 4, 1}, {5, 8, 1}}
shapes[5] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[6] = {2, 0, {5, 9, 1}, {-2, -1, 1}}
```

```
shapes[7] = {2, 1, {10, 10, 1}, {-2, -2, 1}}
shapes[8] = {2, 1, {8, 8, 1}, {3, 3, 1}}
shapes[9] = {2, 1, {5, 7, 1}, {5, 12, 1}}
shapes[10] = {2, 0, {5, 8, 1}, {0, 2, 1}}
shapes[11] = {2, 1, {9, 9, 1}, {0, 0, 1}}
shapes[12] = {2, 1, {5, 7, 1}, {3, 4, 1}}
shapes[13] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Resultado: Incorrecto
```

Listing 5.3: Salidas esperadas y obtenidas por la unión para el caso 1.

Como se pueda apreciar en la comparativa entre las salidas esperadas y obtenidas 5.15, los cálculos realizados por el algoritmo de unión son incorrectos para la entrada del caso de prueba 1. En la subsección 5.1.3 se describe el proceso para determinar el origen del error y las modificiaciones realizadas en el algoritmo para obtener la salida esperada y asegurar la validez del algoritmo.

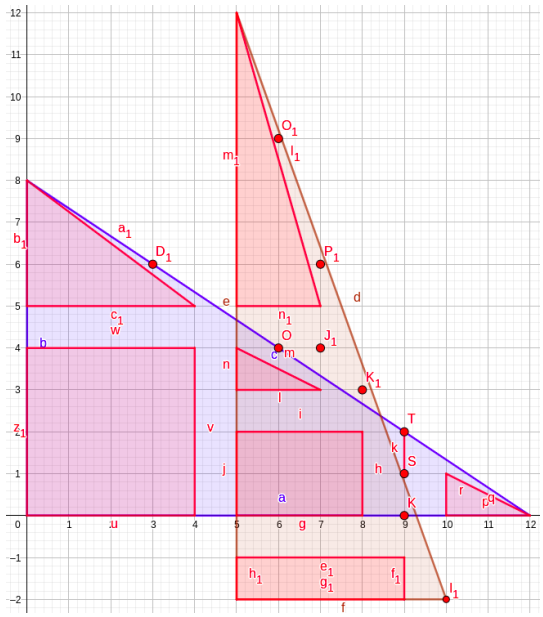

Figura 5.4: Salida esperada del caso de prueba 1 para el algoritmo de unión.

- **Caso de prueba 2**

  Dos triángulos como entrada que determinan un polígono de intersección cuyos vértices tienen más de un índice de distancia entre ellos, que con el reajuste degenera en un segmento.

  - **Shapes de entrada**

    $Sa = \{2, 1, \{0, 12, 1\}, \{0, 8, 1\}\}$

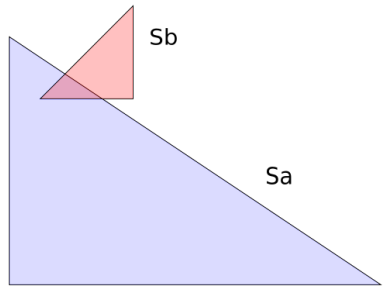

Figura 5.5: Caso de prueba 2.

$$Sb = \{2, 1, \{4, 1, 1\}, \{6, 9, 1\}\}$$

- **Salida del caso de prueba 2 para el algoritmo de intersección**

```
Caso 2 - intersection

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {4, 1, 1}, {6, 9, 1}}

Shapes esperados:
shapes[0] = {2, 1, {1, 3, 1}, {6, 6, 1}}

Shapes obtenidos:
shapes[0] = {2, 1, {1, 3, 1}, {6, 6, 1}}

Resultado: Correcto
```

Listing 5.4: Salidas esperadas y obtenidas por la interescción para el caso 2.

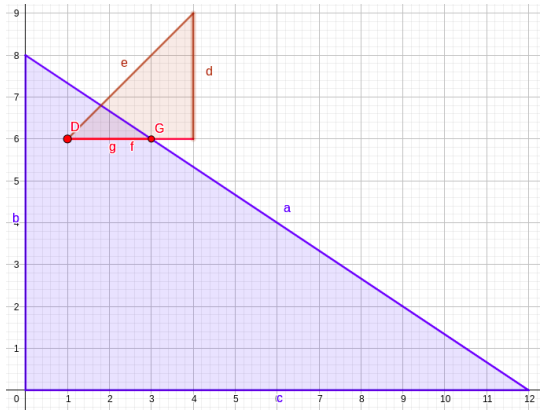

Figura 5.6: Salida del caso de prueba 2 para el algoritmo de intersección.

- **Salida del caso de prueba 2 para el algoritmo de la diferencia**

```
Caso 1 - difference
```

```
Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {5, 10, 1}, {-2, 12, 1}}

Shapes esperados:
shapes[0] = {2, 0, {0, 2, 1}, {0, 5, 1}}
shapes[1] = {2, 1, {3, 12, 1}, {0, 5, 1}}
shapes[2] = {2, 1, {0, 1, 1}, {7, 8, 1}}
shapes[3] = {2, 0, {0, 0, 1}, {6, 6, 1}}
shapes[4] = {2, 0, {4, 4, 1}, {5, 5, 1}}
shapes[5] = {2, 0, {5, 5, 1}, {4, 4, 1}}
shapes[6] = {2, 0, {6, 6, 1}, {4, 4, 1}}
shapes[7] = {2, 0, {7, 7, 1}, {3, 3, 1}}
shapes[8] = {2, 0, {9, 9, 1}, {2, 2, 1}}

Shapes obtenidos:
shapes[0] = {2, 0, {0, 2, 1}, {0, 5, 1}}
shapes[1] = {2, 1, {3, 12, 1}, {0, 5, 1}}
shapes[2] = {2, 1, {0, 1, 1}, {7, 8, 1}}
shapes[3] = {2, 0, {0, 0, 1}, {6, 6, 1}}
shapes[4] = {2, 0, {4, 4, 1}, {5, 5, 1}}
shapes[5] = {2, 0, {5, 5, 1}, {4, 4, 1}}
shapes[6] = {2, 0, {6, 6, 1}, {4, 4, 1}}
shapes[7] = {2, 0, {7, 7, 1}, {3, 3, 1}}
shapes[8] = {2, 0, {9, 9, 1}, {2, 2, 1}}

Resultado: Correcto
```

Listing 5.5: Salidas esperadas y obtenidas por la diferencia para el caso 2.

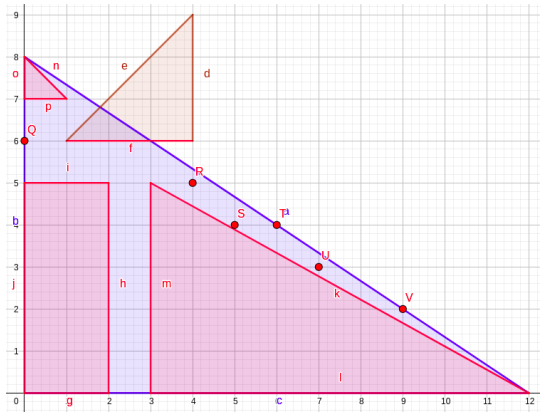

Figura 5.7: Salida del caso de prueba 2 para el algoritmo de la diferencia.

- **Salida del caso de prueba 2 para el algoritmo de unión**

```
Caso 1 - unionHitSig

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {4, 1, 1}, {6, 9, 1}}
```

```
Shapes esperados:
shapes[0] = {2, 1, {3, 12, 1}, {0, 5, 1}}
shapes[1] = {2, 1, {0, 1, 1}, {7, 8, 1}}
shapes[2] = {2, 0, {0, 4, 1}, {6, 6, 1}}
shapes[3] = {2, 1, {4, 4, 1}, {5, 5, 1}}
shapes[4] = {2, 1, {5, 6, 1}, {4, 4, 1}}
shapes[5] = {2, 1, {7, 7, 1}, {3, 3, 1}}
shapes[6] = {2, 0, {9, 9, 1}, {2, 2, 1}}
shapes[7] = {2, 1, {4, 2, 1}, {7, 9, 1}}
shapes[8] = {2, 1, {0, 1, 1}, {0, 4, 1}}
shapes[9] = {2, 1, {2, 0, 1}, {5, 0, 1}}
shapes[10] = {2, 1, {1, 1, 1}, {1, 2, 1}}

Shapes obtenidos:
shapes[0] = {2, 1, {3, 12, 1}, {0, 5, 1}}
shapes[1] = {2, 1, {0, 1, 1}, {7, 8, 1}}
shapes[2] = {2, 0, {0, 4, 1}, {6, 6, 1}}
shapes[3] = {2, 1, {4, 4, 1}, {5, 5, 1}}
shapes[4] = {2, 1, {5, 6, 1}, {4, 4, 1}}
shapes[5] = {2, 1, {7, 7, 1}, {3, 3, 1}}
shapes[6] = {2, 0, {9, 9, 1}, {2, 2, 1}}
shapes[7] = {2, 1, {4, 2, 1}, {7, 9, 1}}
shapes[8] = {2, 1, {0, 1, 1}, {0, 4, 1}}
shapes[9] = {2, 1, {2, 0, 1}, {5, 0, 1}}
shapes[10] = {2, 1, {1, 1, 1}, {1, 2, 1}}

Resultado: Correcto
```

Listing 5.6: Salidas esperadas y obtenidas por la unión para el caso 2.

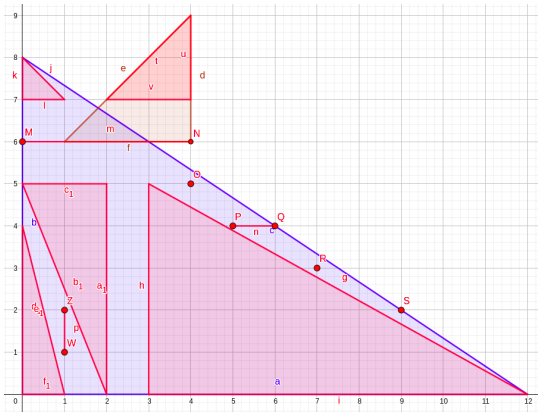

Figura 5.8: Salida del caso de prueba 2 para el algoritmo de unión.

5.1.2. Casos límite

Los casos límite [36] de un conjuntos de pruebas de validación son aquellos que buscan comprobar la corrección del funcionamiento de un software para un conjunto de entradas con valores extremos. Aplicado al contexto de este proyecto se tratan de aquellas entradas que

dan lugar a situaciones complejas entre *shapes*. Esta práctica se basa en el hecho heurístico de que los errores tienden a producirse con mayor probabilidad en los valores extremos de los posibles valores de entrada.

- **Caso de prueba 3**

  Dos triángulos como entrada con un único punto en común.

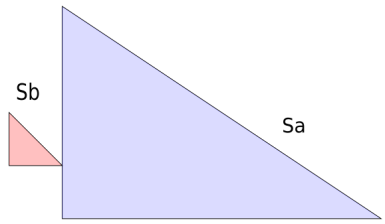

Figura 5.9: Caso de prueba 3.

- **Shapes de entrada**

  $Sa = \{2, 1, \{0, 12, 1\}, \{0, 8, 1\}\}$
  $Sb = \{2, 1, \{-2, 0, 1\}, \{2, 4, 1\}\}$

- **Salida del caso de prueba 3 para el algoritmo de intersección**

```
Caso 3 - intersection

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {-2, 0, 1}, {2, 4, 1}}

Shapes esperados:
shapes[0] = {2, 1, {0, 0, 1}, {2, 2, 1}}


Shapes obtenidos:
shapes[0] = {2, 1, {0, 0, 1}, {2, 2, 1}}


Resultado: Correcto
```

Listing 5.7: Salidas esperadas y obtenidas por la intersección para el caso 3.

- **Salida del caso de prueba 3 para el algoritmo de la diferencia**

```
Caso 3 - difference

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {-2, 0, 1}, {2, 4, 1}}

Shapes esperados:
shapes[0] = {2, 0, {0, 8, 1}, {0, 1, 1}}
```

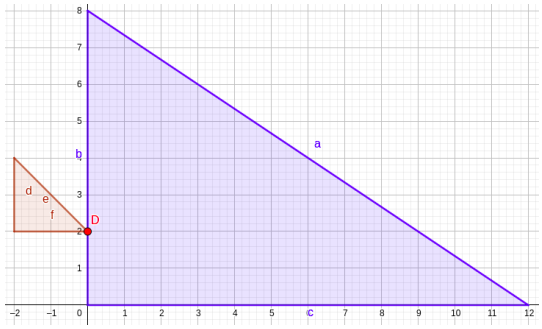

Figura 5.10: Salida del caso de prueba 3 para el algoritmo de intersección.

```
shapes[1] = {2, 1, {9, 12, 1}, {0, 1, 1}}
shapes[2] = {2, 1, {1, 9, 1}, {2, 7, 1}}
shapes[3] = {2, 1, {10, 10, 1}, {1, 1, 1}}
shapes[4] = {2, 0, {0, 0, 1}, {3, 8, 1}}
shapes[5] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[6] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Shapes obtenidos:
shapes[0] = {2, 0, {0, 8, 1}, {0, 1, 1}}
shapes[1] = {2, 1, {9, 12, 1}, {0, 1, 1}}
shapes[2] = {2, 1, {1, 9, 1}, {2, 7, 1}}
shapes[3] = {2, 1, {10, 10, 1}, {1, 1, 1}}
shapes[4] = {2, 0, {0, 0, 1}, {3, 8, 1}}
shapes[5] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[6] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Resultado: Correcto
```

Listing 5.8: Salidas esperadas y obtenidas por la diferencia para el caso 3.

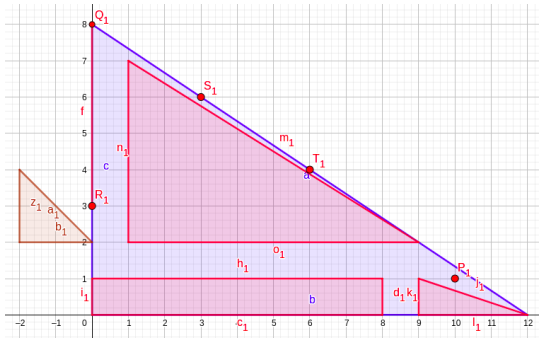

Figura 5.11: Salida del caso de prueba 3 para el algoritmo de la diferencia.

- **Salida del caso de prueba 3 para el algoritmo de unión**

```
Caso 3 - unionHitSig

Entrada:
```

```
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {5, 10, 1}, {-2, 12, 1}}

Shapes esperados:
shapes[0] = {2, 1, {0, 12, 1}, {0, 8, 1}}
shapes[1] = {2, 1, {-2, -1, 1}, {2, 4, 1}}
shapes[2] = {2, 0, {-1, -1, 1}, {3, 3, 1}}



Shapes obtenidos:
shapes[0] = {2, 1, {0, 12, 1}, {0, 8, 1}}
shapes[1] = {2, 1, {-2, -1, 1}, {2, 4, 1}}
shapes[2] = {2, 0, {-1, -1, 1}, {3, 3, 1}}

Resultado: Correcto
```

Listing 5.9: Salidas esperadas y obtenidas por la unión para el caso 3.

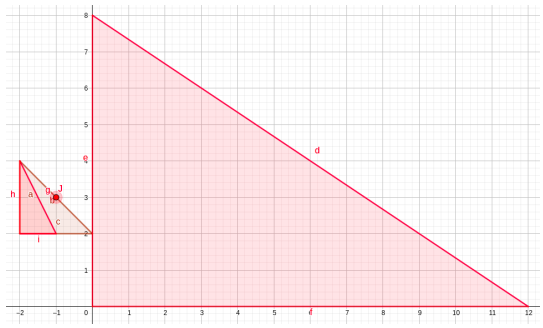

Figura 5.12: Salida del caso de prueba 3 para el algoritmo de unión.

- **Caso de prueba 4**

  Dos triángulos como entrada cuya intersección es un segmento diagonal.

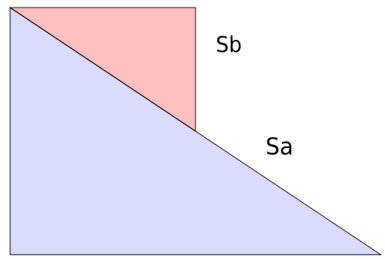

Figura 5.13: Caso de prueba 4.

- **Shapes de entrada**

  $Sa = \{2, 1, \{0, 12, 1\}, \{0, 8, 1\}\}$

$Sb = \{2, 1, \{6, 0, 1\}, \{8, 4, 1\}\}$

- **Salida del caso de prueba 4 para el algoritmo de intersección**

```
Caso 4 - intersection

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {6, 0, 1}, {8, 4, 1}}

Shapes esperados:
shapes[0] = {2, 1, {0, 0, 1}, {8, 8, 1}}
shapes[1] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[2] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Shapes obtenidos:
shapes[0] = {2, 1, {0, 0, 1}, {8, 8, 1}}
shapes[1] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[2] = {2, 1, {6, 6, 1}, {4, 4, 1}}

Resultado: Correcto
```

Listing 5.10: Salidas esperadas y obtenidas por la intersección para el caso 4.

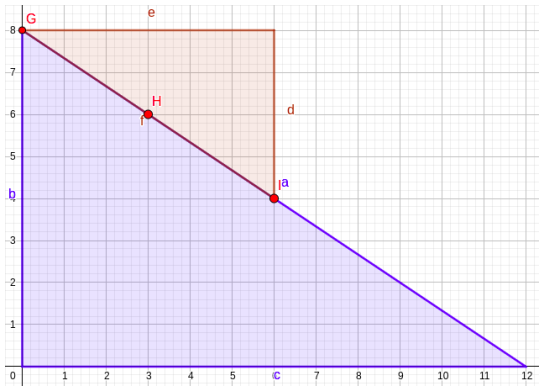

Figura 5.14: Salida del caso de prueba 4 para el algoritmo de intersección.

- **Salida del caso de prueba 4 para el algoritmo de la diferencia**

```
Caso 4 - difference

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {6, 0, 1}, {8, 4, 1}}

Shapes esperados:
shapes[0] = {2, 0, {0, 5, 1}, {0, 3, 1}}
shapes[1] = {2, 1, {6, 12, 1}, {0, 3, 1}}
shapes[2] = {2, 1, {0, 5, 1}, {4, 7, 1}}
shapes[3] = {2, 1, {7, 7, 1}, {3, 3, 1}}
shapes[4] = {2, 1, {9, 9, 1}, {2, 2, 1}}
shapes[5] = {2, 1, {1, 1, 1}, {7, 7, 1}}
```

```
shapes[6] = {2, 1, {2, 2, 1}, {6, 6, 1}}
shapes[7] = {2, 1, {4, 4, 1}, {5, 5, 1}}

Shapes obtenidos:
shapes[0] = {2, 0, {0, 5, 1}, {0, 3, 1}}
shapes[1] = {2, 1, {6, 12, 1}, {0, 3, 1}}
shapes[2] = {2, 1, {0, 5, 1}, {4, 7, 1}}
shapes[3] = {2, 1, {7, 7, 1}, {3, 3, 1}}
shapes[4] = {2, 1, {9, 9, 1}, {2, 2, 1}}
shapes[5] = {2, 1, {1, 1, 1}, {7, 7, 1}}
shapes[6] = {2, 1, {2, 2, 1}, {6, 6, 1}}
shapes[7] = {2, 1, {4, 4, 1}, {5, 5, 1}}

Resultado: Correcto
```

Listing 5.11: Salidas esperadas y obtenidas por la diferencia para el caso 4.

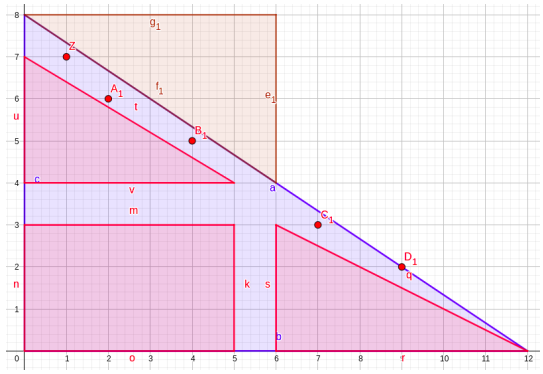

Figura 5.15: Salida del caso de prueba 4 para el algoritmo de la diferencia.

- **Salida del caso de prueba 4 para el algoritmo de unión**

```
Caso 4 - unionHitSig

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {6, 0, 1}, {8, 4, 1}}

Shapes esperados:
shapes[0] = {2, 1, {0, 11, 1}, {0, 7, 1}}
shapes[1] = {2, 1, {6, 0, 1}, {8, 4, 1}}
shapes[2] = {2, 0, {12, 12, 1}, {0, 0, 1}}
shapes[3] = {2, 1, {1, 1, 1}, {7, 7, 1}}
shapes[4] = {2, 1, {2, 2, 1}, {6, 6, 1}}
shapes[5] = {2, 1, {4, 4, 1}, {5, 5, 1}}
shapes[6] = {2, 1, {5, 5, 1}, {4, 4, 1}}
shapes[7] = {2, 1, {7, 7, 1}, {3, 3, 1}}
shapes[8] = {2, 1, {8, 9, 1}, {2, 2, 1}}
shapes[9] = {2, 1, {10, 10, 1}, {1, 1, 1}}

Shapes obtenidos:
shapes[0] = {2, 1, {0, 11, 1}, {0, 7, 1}}
shapes[1] = {2, 1, {6, 0, 1}, {8, 4, 1}}
shapes[2] = {2, 0, {12, 12, 1}, {0, 0, 1}}
```

```
shapes[3] = {2, 1, {1, 1, 1}, {7, 7, 1}}
shapes[4] = {2, 1, {2, 2, 1}, {6, 6, 1}}
shapes[5] = {2, 1, {4, 4, 1}, {5, 5, 1}}
shapes[6] = {2, 1, {5, 5, 1}, {4, 4, 1}}
shapes[7] = {2, 1, {7, 7, 1}, {3, 3, 1}}
shapes[8] = {2, 1, {8, 9, 1}, {2, 2, 1}}
shapes[9] = {2, 1, {10, 10, 1}, {1, 1, 1}}

Resultado: Correcto
```

Listing 5.12: Salidas esperadas y obtenidas por la unión para el caso 4.

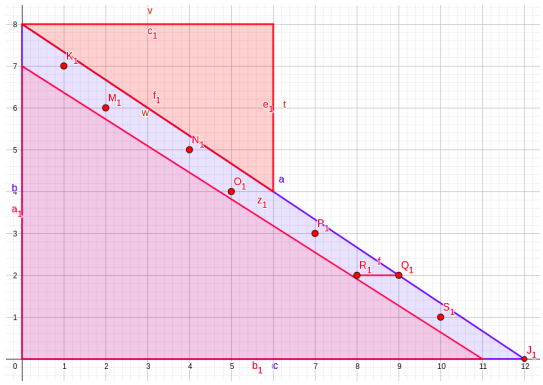

Figura 5.16: Salida del caso de prueba 4 para el algoritmo de unión.

- **Caso de prueba 5**

  Dos triángulos como entrada que no presentan ningún punto común.

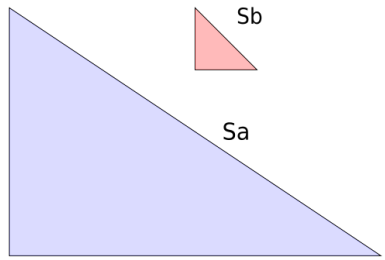

Figura 5.17: Caso de prueba 5.

- **Shapes de entrada**

  $Sa = \{2, 1, \{0, 12, 1\}, \{0, 8, 1\}\}$
  $Sb = \{2, 1, \{5, 10, 1\}, \{-2, 12, 1\}\}$

- **Salida del caso de prueba 1 para el algoritmo de intersección**

```
Caso 5 - intersection

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {6, 8, 1}, {6, 8, 1}}

Shapes esperados:

Shapes obtenidos:

Resultado: Correcto
```

Listing 5.13: Salidas esperadas y obtenidas por la intersección para el caso 5.

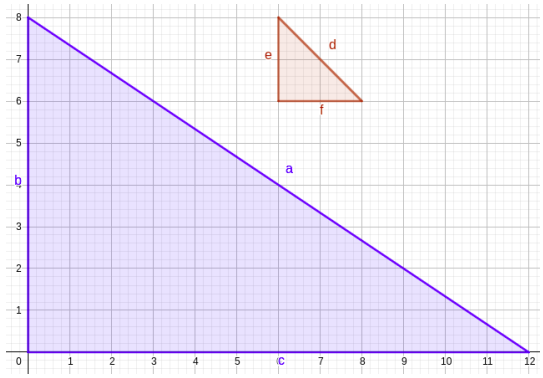

Figura 5.18: Salida del caso de prueba 5 para el algoritmo de intersección.

- **Salida del caso de prueba 5 para el algoritmo de la diferencia**

```
Caso 5 - difference

Entrada:
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {6, 8, 1}, {6, 8, 1}}

Shapes esperados:
shapes[0] = {2, 1, {0, 12, 1}, {0, 8, 1}}


Shapes obtenidos:
shapes[0] = {2, 1, {0, 12, 1}, {0, 8, 1}}


Resultado: Correcto
```

Listing 5.14: Salidas esperadas y obtenidas por la diferencia para el caso 5.

- **Salida del caso de prueba 5 para el algoritmo de unión**

```
Caso 5 - unionHitSig

Entrada:
```

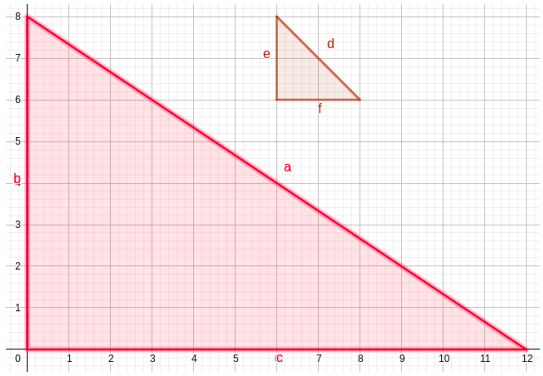

Figura 5.19: Salida del caso de prueba 1 para el algoritmo de la diferencia.

```
Sa = {2, 1, {0, 12, 1}, {0, 8, 1}}
Sb = {2, 1, {5, 10, 1}, {-2, 12, 1}}

Shapes esperados:
shapes[0] = {2, 1, {0, 12, 1}, {0, 8, 1}}
shapes[1] = {2, 1, {6, 8, 1}, {6, 8, 1}}

Shapes obtenidos:
shapes[0] = {2, 1, {0, 12, 1}, {0, 8, 1}}
shapes[1] = {2, 1, {6, 8, 1}, {6, 8, 1}}

Resultado: Correcto
```

Listing 5.15: Salidas esperadas y obtenidas por la unión para el caso 1.

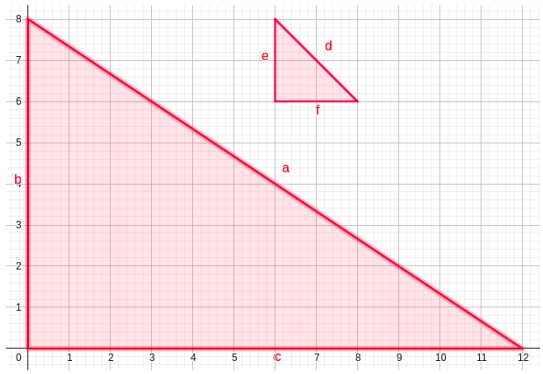

Figura 5.20: Salida esperada del caso de prueba 1 para el algoritmo de unión.

5.1.3. Correciones de los algoritmos

Como se ha visto en la sección 5.1.1, el algoritmo de unión produce una salida incorrecta para el caso de prueba 1. Este caso de prueba consiste en las siguientes entradas:

Shapes de entrada

$Sa = \{2, 1, \{0, 12, 1\}, \{0, 8, 1\}\}$
$Sb = \{2, 1, \{5, 10, 1\}, \{-2, 12, 1\}\}$

Para estas entradas, la salida esperada para el algoritmo de unión es la siguiente:

```
shapes[0] = {2, 0, {0, 4, 1}, {0, 4, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {1, 2, 1}}
shapes[2] = {2, 1, {10, 12, 1}, {0, 1, 1}}
shapes[3] = {2, 1, {0, 4, 1}, {5, 8, 1}}
shapes[4] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[5] = {2, 0, {5, 9, 1}, {−2, −1, 1}}
shapes[6] = {2, 1, {10, 10, 1}, {−2, −2, 1}}
shapes[7] = {2, 1, {7, 7, 1}, {4, 4, 1}}
shapes[8] = {2, 1, {8, 8, 1}, {3, 3, 1}}
shapes[9] = {2, 1, {5, 7, 1}, {5, 12, 1}}
shapes[10] = {2, 1, {6, 6, 1}, {9, 9, 1}}
shapes[11] = {2, 1, {7, 7, 1}, {6, 6, 1}}
shapes[12] = {2, 0, {5, 8, 1}, {0, 2, 1}}
shapes[13] = {2, 1, {9, 9, 1}, {0, 0, 1}}
shapes[14] = {2, 1, {5, 7, 1}, {3, 4, 1}}
shapes[15] = {2, 1, {6, 6, 1}, {4, 4, 1}}
```

Listing 5.16: Salida obtenida por la unión para el caso 1.

Que se corresponde con la siguiente figura:

Sin embargo, la salida obtenida por el algoritmo de unión es la siguiente:

```
shapes[0] = {2, 0, {0, 4, 1}, {0, 4, 1}}
shapes[1] = {2, 1, {9, 9, 1}, {1, 1, 1}}
shapes[2] = {2, 1, {10, 12, 1}, {0, 1, 1}}
shapes[3] = {2, 1, {9, 9, 1}, {2, 2, 1}}
shapes[4] = {2, 1, {0, 4, 1}, {5, 8, 1}}
shapes[5] = {2, 1, {3, 3, 1}, {6, 6, 1}}
shapes[6] = {2, 0, {5, 9, 1}, {−2, −1, 1}}
shapes[7] = {2, 1, {10, 10, 1}, {−2, −2, 1}}
shapes[8] = {2, 1, {8, 8, 1}, {3, 3, 1}}
shapes[9] = {2, 1, {5, 7, 1}, {5, 12, 1}}
shapes[10] = {2, 0, {5, 8, 1}, {0, 2, 1}}
shapes[11] = {2, 1, {9, 9, 1}, {0, 0, 1}}
shapes[12] = {2, 1, {5, 7, 1}, {3, 4, 1}}
shapes[13] = {2, 1, {6, 6, 1}, {4, 4, 1}}
```

Listing 5.17: Salidas obtenidas por la unión para el caso 1.

Que se corresponde con la siguiente figura:

Como se pueda apreciar en la figura 5.22, se produce la pérdida de los puntos con coordenadas (7 , 4), (6 , 9) y (7 , 6), que se corresponden con los shapes 7, 10 y 11 del listing 5.16, respectivamente.

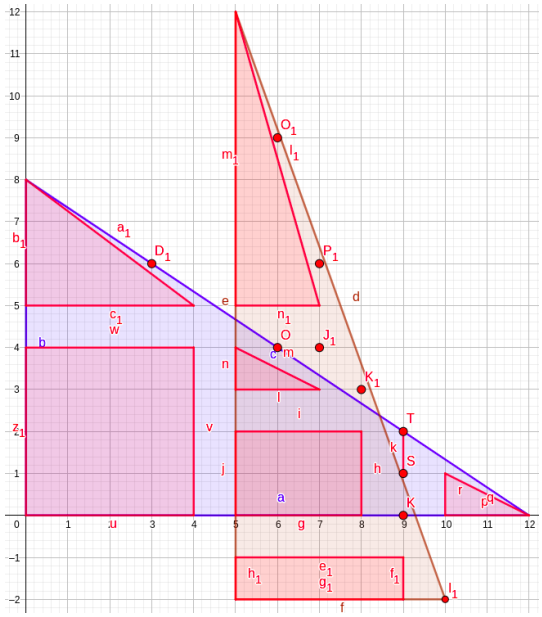

Figura 5.21: Salida esperada del caso de prueba 1 para el algoritmo de unión.

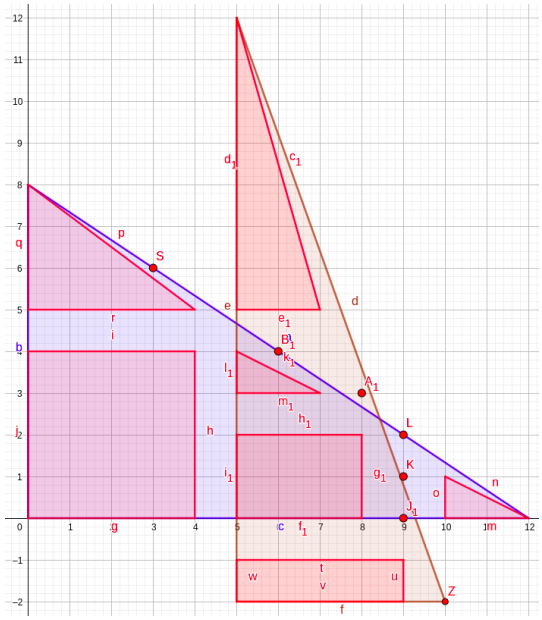

Figura 5.22: Salida obtenida del caso de prueba 1 para el algoritmo de unión.

Observando las salidas se puede apreciar que todos los shapes perdidos pertenecen a la diferencia de Sb menos Sa. Como se ha descrito en la sección de 3.5.3, el algoritmo de intersección realiza el cálculo de la intersección de Sa con Sb, la diferencia de Sa menos Sb y

la diferencia de Sb menos Sa. Esto acota el error de los cálculos al algoritmo de la diferencia, en concreto para una situación generada para la diferencia de Sb menos Sa. Estos resultados chocan con los requisitos RF03, RF04, RNF02 y RNF03 descritos en la sección 3.3, por ende, es necesario realizar modificaciones en el algoritmo para lograr la corrección de los cálculos y cumplir con todos los requisitos funcionales y no funcionales.

Tras una fase de debug se localiza el punto concreto en el que se origina el error: el reajuste de un triángulo de coordenadas reales a enteras y su posterior generación de puntos outsiders. Este triángulo se trata del shape 9 del array de salida, el cual se trata de un triángulo de tipo 1. La problemática surge al aplicar un reajuste que no se corresponde con el tipo de triángulo que se trata, por lo que se crea una función denominada hit_sig_ext_shape_share_side_part que detecta ciertas condiciones particulares que distinguen el tipo de triángulo y coincidencias con shapes contiguos, de forma que se invoca a la función de reajuste correcta.

La búsqueda del origen del error de las salidas en este caso de prueba y la posterior fase de modificación del software para lograr la validez del algoritmo ha afectado ligeramente al camino crítico. Esta tarea de depuración costo dos días completos de trabajo que obligaron a reducir el tiempo empleado para desarrollar esta memoria con el objetivo de alcanzar la fecha límite de finalización del proyecto.

Capítulo 6

Conclusiones

En este capítulo se introducen los siguientes aspectos:

- Los objetivos del proyecto que se han sido completados.
- Las líneas de trabajo a realizar a raíz de este proyecto.
- Una opinión personal sobre el desarrollo del proyecto.

6.1. Objetivos cumplidos

En este proyecto fin de grado se ha desarrollado dentro de un proyecto del grupo de investigación Trasgo para permitir el uso de n-símplices (estructuras triangulares) con lados paralelos a los ejes de coordenadas a la hora de particionar el espacio de memoria, partiendo de la estructura de datos HitSigShape, analizada en la sección 3.1 y que sirve como extensión de la librería Hitmap,.

Tras el desarrollo del proyecto se ha logrado conseguir la siguientes características:

- El software está correctamente documentado.
- El software es extensible y reutilizable.
- El software está bien estructurado en subfunciones, con el fin de evitar duplicidad de código, y facilita el seguimiento del flujo del programa.
- El software utiliza la notación y los paradigmas de programación del grupo Trasgo.
- El software está bien preparado para cambios futuros.
- El software es correcto en sus cálculos.

Además, para el desarrollo de este proyectado se han realizado las siguientes tareas y subtareas:

- Se ha estudiado, analizado y comprendido las principales estructuras de Hitmap.
- Se ha estudiado, analizado y comprendido las operaciones binarias de intersección, unión y diferencia entre dos polígonos para un espacio de dos dimensiones.
- Se ha diseñado e implementado las estructuras de datos extendidas de la librería Hitmap para poder modelar particiones con formas triangulares.
- Se ha diseñado e implementado la operación de intersección entre dos rectángulos, dos triángulos o un triángulo y un rectángulo.
- Se ha diseñado e implementado la operación de unión entre dos rectángulos, dos triángulos o un triángulo y un rectángulo.
- Se ha diseñado e implementado la operación de diferencia entre dos rectángulos. dos triángulos o un triángulo y un rectángulo.
- Se ha diseñado, elaborado y ejecutado un plan de pruebas que valide la corrección de los cálculos realizados por los algoritmos.
- Se ha analizado los resultados y solventado aquellos errores surgidos de las pruebas de validación.
- Se ha elaborado un documento que refleja de forma clara y precisa la naturaleza del proyecto, su estructura, el proceso seguido y las conclusiones extraídas.

Este Trabajo de Fin de Grado da lugar a un artículo científico, en el que figuro como autor, publicado en la *XXI International Computational and Mathematical Methods in Science and Engineering* [37], que se celebrará en Salamanca, del 22 al 27 de julio del 2021. En este paper, que tiene por título "Triangular Tilling for reducing communications in Hitmap library".

6.2. Trabajo futuro

Tras la implementación de la extensión de la librería Hitmap del grupo de investigación Trasgo para la triangularización del espacio de datos surgen una serie de líneas de trabajo futuro.

La primera línea de trabajo es el desarrollo de la librería Hitmap para permitir la ejecución de aplicaciones en entornos heterogéneos. Se estudiarán y analizarán benchmarks y aplicaciones reales científicas apropiadas que aprovechen las características de este tipo de entornos.

Como se ha descrito en los capítulos 3 y 4, las operaciones implementadas para trabajar con las particiones triangulares de memoria son la intersección, diferencia y unión de dos

shapes. Estas operaciones pueden ser extendidas para poder ser realizadas sobre un conjunto de polígonos, de forma que, para cada uno de los tres algoritmos se reciba una número variable de shapes como entrada y obtener la intersección o unión de todos con todos o la diferencia del primero menos los restantes. Esto permitiría alcanzar un grado superior de optimización en el particionado de espacios de datos con geometrías irregulares.

El trabajo desarrollado para el particionado triangular puede ser extendido para trabajar con un número mayor de dimensiones. Actualmente, la extensión de Hitmap trabaja en 2 dimensiones creando shapes que representan triángulos. Al extenderse para 3 dimensiones, las particiones de memoria podrían adquirir formas como prismas triangulares o pirámides, facilitando el trabajo para espacios de memoria cada vez más complejos. Esta extensión podría realizarse incluso para n dimensiones.

6.3. Valoración personal

Este Trabajo de Fin de Grado, al encuadrarse dentro de un proyecto de investigación, ha resultado ser una experiencia muy diferente del desarrollo software convencional, con el cuál he estado más familiarizado debido a la orientación del grado en Ingeniería Informática de la UVa. He descubierto el área de la informática de la investigación científica y he entendido mejor el funcionamiento de la empresa TIC dentro de la investigación pública y del sector privado.

Es satisfactorio comprobar que los objetivos fijados durante el planteamiento inicial del proyecto han sido cumplidos con éxito y dentro de las fechas límite fijadas por la planificación. Como consecuencia de esto, me ha sido posible entregar esta memoria junto a la extensión de la librería en la convocatoria ordinaria del curso académico 2020/2021.

Durante algo más de un cuatrimestre, he trabajado con el grupo de investigación Trasgo del departamento de informática de la Universidad de Valladolid donde he ampliado mis conocimientos sobre la computación paralela que había adquirido anteriormente en la asignatura CPAR del este mismo grado. A mayores, he trabajado con nuevos Sistemas Operativos, que me han permitido entender mejor como usan los recursos subyacentes de las máquinas heterogéneas de computación paralela del grupo Trasgo. Durante la programación de este proyecto he desarrollado mi destreza en la programación con el lenguaje C y además he conocido y practicado nuevas metodologías de trabajo, propias de la investigación y mas alejadas del desarrollo software clásico.

Capítulo 7

apéndices

Apéndice A

Contenidos del proyecto

La entrega de este Trabajo de Fin de Grado ha sido realizada a través del fichero ZIP DatosTFGHugoPrietoTarrega, que contiene los siguientes archivos:

- **intersecion.h:** fichero de cabecera [38] que incluye la declaración de las estructuras de datos, macros y funciones empleadas por los algoritmos que extienden la librería Hitmap.
- **intersection.c:** fichero que contiene el código fuente de la extensión la librería Hitmap desarrollada en este.

Bibliografía

[1] G. Hager y G. Wellein. *Introduction to high performance computing for scientists and engi- neers*. CRC Press, 2010.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[3] Ivan S. Ufimtsev and Todd J. Martínez. Graphical processing units for quantum chemistry. *Computing in Science Engineering*, 10(6):26–34, 2008.

[4] Sparsh Mittal. A Survey of FPGA-based Accelerators for Convolutional Neural Networks. Febrero 2020. `https://www.researchgate.net/publication/327931012_A_Survey_of_FPGA-based_Accelerators_for_Convolutional_Neural_Networks`.

[5] Jorge Ortíz. La computación paralela: alta capacidad de procesamiento, 2020. `https://www.teldat.com/blog/es/computacion-paralela-capacidad-procesamiento/`, Último acceso: Junio, 2021.

[6] TOP 500. TOP 500 - HomePage , 2021. `https://www.top500.org/`, Último acceso: Junio, 2021.

[7] Felipe Restrepo Calle. Ventajas y desventajas de la computación paralela. `http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html#four`, Último acceso: Junio, 2021.

[8] TOP 500. HIGHLIGHTS - NOVEMBER 2020, noviembre 2020. `https://www.top500.org/`, Último acceso: Junio, 2021.

[9] Juan Antonio Pascual Estapé. Petaflops, la unidad de medida de los superordenadores, julio 2020. `https://computerhoy.com/reportajes/tecnologia/petaflops-unidad-medida-superordenadores-667982`, Último acceso: Junio, 2021.

[10] Juan Carlos López. Con MareNostrum 5 podríamos quedar entre los tres supercomputadores más rápidos del mundo, pero no es nuestro objetivo», Mateo Valero, director del BSC, Febrero 2020. `https://www.xataka.com/investigacion/marenostrum-5-podriamos-quedar-tres-supercomputadores-rapidos-mundo-no-nuestro-o` Último acceso: Junio, 2021.

[11] Eloy Fustero. ¿Qué es la computación heterogénea? Parte II, Noviembre 2013. `https://blogthinkbig.com/computacion-heterogenea-2`, Último acceso: Junio, 2021.

[12] Arturo González-Escribano. Grupo Trasgo, 2020. `https://trasgo.infor.uva.es/`, Último acceso: Junio, 2021.

[13] UVa. Departamento de Informática de la Universidad de Valladolid, 2021. `https://www.infor.uva.es/`, Último acceso: Junio, 2021.

[14] Javier Fresno Diego R. Llanos Arturo Gonzalez-Escribano, Yuri Torres. An Extensible System for Multilevel Automatic Data Partition and Mapping. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1–2, 2014.

[15] W. Stallings. *Computer organization and architecture: Designing for performance.* Upper Saddle River, NJ: Prentice Hall. 2010.

[16] Jonathan Sterne. Plug-in. *Encyclopedia Britannica*, Octubre 2019. `https://www.britannica.com/technology/plug-in`. Último acceso: junio 2021.

[17] Tech Terms. Software Terms : Framework Definition, March 2013. `shorturl.at/firFJ`, Último acceso: Junio, 2021.

[18] Yuri Torres, Arturo Gonzalez-Escribano, and Diego Ferraris. Encapsulated synchronization and load-balance in heterogeneous programming. pages 502–513, 08 2012.

[19] Guía Docente Computación Paralela Curso 2018-2019, 2018.

[20] Diego R. Llanos Ana Moreton-Fernandez, Arturo Gonzalez-Escribano. A technique to automatically determine Ad-hoc communication patterns at runtime. *Parallel Computing 69*, pages 45 – 62, 2017.

[21] Arturo González Escribano María Inmaculada Santamaría Valenzuela, Yuri Torres de la Sierra. Triangular tiling. an extension for hitmap's hitsigshape structure and its algebra, (pendiente de envío). pages 1–6, 2021.

[22] CMS. Selecting a development approach . pages 1 – 10, marzo 2008.

[23] Bob Hughes and Mike Cotterel. Software project management. pages 49–71, May 2009.

[24] Discord. Discord: Main Page , 2021. `https://discord.com/brand-new`, Último acceso: Junio, 2021.

[25] Bob Hughes and Mike Cotterel. Software project management. pages 162–188, May 2009.

[26] Bob Hughes and Mike Cotterel. Software project management. pages 129–160, May 2009.

[27] Encyclopedia of Mathematics. Parallelotope, 2020. `https://encyclopediaofmath.org/wiki/Parallelotope`, Último acceso: Junio, 2021.

[28] Técnicas para Identificar Requisitos Funcionales y No Funcionales, 2020. `https://sites.google.com/site/metodologiareq/capitulo-ii/tecnicas-para-identificar-requisitos-funcionales-y-no-funcionales`.

[29] Robert A. Beezer Thomas W. Judson. Álgebra abstracta, teoría y aplicaciones. pages 1–14, 2017.

[30] Alberto Márquez. Intersección de polígonos convexos, 2020. `https://n9.cl/lo8yu`, Último acceso: Junio, 2021.

[31] Erich Hartmann. Geometry and algorithms for computer aided design. page 17, 2003.

[32] Arturo González Escribano María Inmaculada Santamaría Valenzuela, Yuri Torres de la Sierra. Triangulartiling. an extension for hitmap's hitsigshape structure and its algebra. pages 6–29, 2021.

[33] tutorials point. C library function - calloc(), 2017. `https://www.tutorialspoint.com/c_standard_library/c_function_calloc.htm`.

[34] Microsoft. Por qué los números de punto flotante pierden precisión, 2016. `https://docs.microsoft.com/es-es/cpp/build/why-floating-point-numbers-may-lose-precision?view=msvc-160`.

[35] Microsoft. Precisión y precisión en los cálculos de punto flotante, 2020. `https://docs.microsoft.com/es-es/office/troubleshoot/access/floating-calculations-info`.

[36] Educando con TIC. Valores límite: Pruebas de caja negra, 2021. `https://educandocontic.com/valores-limite-pruebas/`, Último acceso: Junio, 2021.

[37] Univeridad de Salamanca. 21 International Conference Computational and Mathematical Methods in Science and Engineering, 2021. `https://cmmse.usal.es/cmmse2021/welcome/`.

[38] Wikipedia. Archivo de cabecera, 2021. `https://es.wikipedia.org/wiki/Archivo_de_cabecera`.