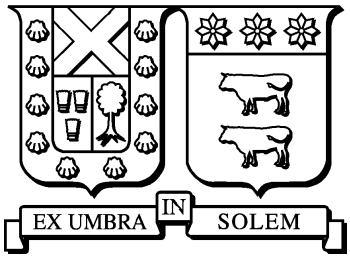


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

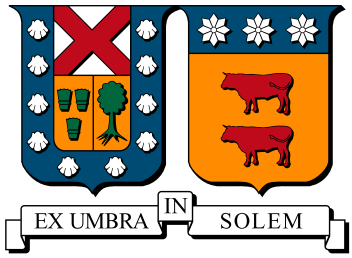
DEPARTAMENTO DE INFORMÁTICA

ESTRUCTURA DE DATOS BASADA EN NODOS PARA LA OPTIMIZACIÓN DEL PROCESO DE BALANCEO EN MALLAS GEOMÉTRICAS TIPO QUADTREE/OCTREE.

Jorge Ariel Díaz Matte

MAGÍSTER EN CIENCIAS DE LA INGENIERÍA INFORMÁTICA

ENERO 2024



UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA

ESTRUCTURA DE DATOS BASADA EN NODOS PARA LA OPTIMIZACIÓN DEL PROCESO DE BALANCEO EN MALLAS GEOMÉTRICAS TIPO QUADTREE/OCTREE.

Tesis de Grado presentada por
JORGE ARIEL DÍAZ MATTE

como requisito parcial para optar al grado de
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA INFORMÁTICA

PROFESOR GUÍA
MAURICIO SOLAR

ENERO 2024

TÍTULO DE LA TESIS:

ESTRUCTURA DE DATOS BASADA EN NODOS PARA LA OPTIMIZACIÓN DEL PROCESO DE BALANCEO EN MALLAS GEOMÉTRICAS TIPO QUADTREE/OCTREE.

AUTOR:

JORGE ARIEL DÍAZ MATTE

Trabajo de Tesis, presentado en cumplimiento parcial de los requisitos para el Grado de **Magíster en Ciencias de la Ingeniería Informática** de la Universidad Técnica Federico Santa María.

Mauricio Solar	Director de Tésis
Claudio Lobos Yañez	Co-Director de Tésis
Nancy Hitschfeld Kahler	Co-Directora de Tésis (U. Chile)
Elizabeth Montero	Correferente Interno
Diego Arroyuelo	Correferente Externo (PUC)
Ricardo Ñanculef	Presidente Comisión

Enero 2024.

Valparaíso, Chile.

Agradecimientos

Agradezco profundamente a mis padres, Nanette y Ariel, por el apoyo y amor que siempre me han dado. A mi pareja, Raiza, por su amor incondicional, aliento y consejos, que me han permitido llegar lejos. A mis hermanos por su preocupación y compañía, y a mi numerosa familia en general.

A mi profesor guía, Claudio Lobos, quien me enseñó, motivó, ayudó y salvó mi permanencia en el programa de postgrado, después de quedarme sin guía en dos ocasiones. A Nancy Hitschfeld, por su apoyo y tiempo en mi trabajo, junto con brindarme oportunidades para seguir investigando y estudiando.

Este trabajo fue parcialmente financiado por el proyecto Fondecyt 1211484.

Resumen

Los métodos numéricos son una técnica matemática utilizada para realizar simulaciones que contribuyen a la resolución de problemas en ingeniería, física, medicina, entre otras disciplinas. Estas técnicas utilizan mallas geométricas para representar los objetos de estudio tanto en 2D como en 3D. Un punto de partida para generar una malla puede ser un *quadtree* o un *octree* dependiendo de la dimensión. Estas estructuras se van dividiendo en subcuadrantes o suboctantes hasta alcanzar el nivel que se requiere. Durante este proceso de refinamiento es posible que dos celdas vecinas pierdan el balance, vale decir, la diferencia entre sus niveles de refinamiento sea mayor a uno.

El balance es necesario en este tipo de mallas ya que se han diseñado un conjunto de patrones para gestionar la transición entre regiones más finas y las más gruesas, produciendo así una malla congruente. Este balance se puede mantener mientras se refina o se puede hacer al terminar el proceso de generación de la malla. En trabajos anteriores se ha demostrado que la primera opción es más rápida para generar la malla. Para lograr esto de forma eficiente es necesario mantener dinámicamente la información sobre los vecinos de cada celda. Actualmente, los algoritmos de mallado logran esto mediante el uso de un enfoque en donde los arcos mantienen la información sobre los vecinos.

En este trabajo, se propone una mejora al proceso de almacenamiento dinámico de la información de las celdas vecinas mediante el diseño de una nueva estructura de datos basada en nodos de la malla, para lograr reducir el tiempo de ejecución y el uso de memoria.

Palabras clave: Mallas Geométricas, Refinamiento de Cuadrantes, Refinamiento de Octantes, Estructura de datos de Nodos.

Abstract

Numerical methods are a mathematical technique used for simulations that contribute to problem-solving in engineering, physics, medicine, among other disciplines. These techniques employ geometric meshes to represent study objects in both 2D and 3D. A starting point for generating a mesh can be either a quadtree or an octree, depending on the dimension. These structures are subdivided into subquadrants or suboctants until the required level is reached. During this refinement process, it's possible for two neighboring cells to become unbalanced, meaning the difference between their refinement levels exceeds one.

Balance is crucial in this type of meshes, as a set of patterns has been designed to manage the transition between finer and coarser regions, thus producing a congruent mesh. This balance can be maintained while refining or formed upon completion. Previous work has demonstrated that the first option is faster for mesh generation. Achieving this efficiently requires dynamically maintaining information about the neighbors of each cell. Currently, meshing algorithms accomplish this through an edge-based approach for neighbor querying

In this work, a improvement is proposed for the storage and dynamic update process of neighboring cell information by designing a new data structure based on mesh nodes instead of edges, aiming to reduce runtime and memory usage.

Keywords: Geometric Meshes, Quadrant Refinement, Octant Refinement, Nodes Data Structure.

Glosario

Quadtree: Estructura de dato de tipo árbol, en donde cada nodo del árbol tiene hasta 4 hijos.

Octree: Estructura de dato de tipo árbol, en donde cada nodo del árbol tiene hasta 8 hijos.

Cuadrante: Unidad básica de un *quadtree* y que geométricamente corresponde a un cuadrado.

Octante: Unidad básica de un *octree* y que geométricamente corresponde a un cubo.

Celda: Término ocupado para referirse indistintamente a un cuadrante u octante.

Refinamiento: Proceso en el que una celda se divide en una cantidad determinada de subceldas iguales. En el caso de los cuadrantes, estos se refinan en cuatro subcuadrantes de igual tamaño, mientras que los octantes se dividen en ocho suboctantes iguales.

Nivel de refinamiento: Número entero que simboliza la cantidad de veces que una celda ha sido refinada hasta llegar a la subcelda que posee dicho nivel de refinamiento.

Polilínea: Es una serie de segmentos de línea consecutivos conectados entre sí, que se utilizan para representar, de manera computacional, formas y contornos complejos.

Malla de superficie: Es una malla geométrica de superficie, compuesta por diferentes polígonos, que define una representación discreta de una superficie en el espacio tridimensional.

Índice de Contenidos

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Un método numérico es una técnica matemática utilizada para realizar simulaciones que contribuyen a la resolución de problemas en ingeniería, física, geología, medicina, entre otros [1]. En estos temas, los objetos de estudio, es decir, el dominio, pueden tener formas no regulares y, por lo tanto, se necesitará una representación discreta. Una forma de hacerlo es generar la llamada malla geométrica. En el caso de dominios 2D, se habla de mallas de superficie, las cuales suelen estar compuestas por triángulos [2] o cuadriláteros [3]. Para los casos de dominios en 3D, están las llamadas mallas volumétricas o mallas superficiales, donde estas últimas son huecas, y las primeras se componenen generalmente de tetraedros [4] o hexaedros [5]. Sin embargo, tanto para las mallas bidimensionales como tridimensionales existen las llamadas mallas de elementos mixtos [6, 7] , las que consisten en mallas que mezclan diferentes tipos de polígonos o poliedros para formar la malla resultante.

Un punto de partida para producir una malla 2D es el llamado *quadtree* [1, 8], que es una estructura de datos jerárquica y recursiva, que se puede representar como un árbol, donde cada nodo del árbol puede tener hasta 4 hijos. De forma análoga, para una malla en tres dimensiones se utiliza el *octree* [9, 10], donde se suele utilizar un árbol equivalente al *quadtree*, pero cada nodo del árbol puede tener hasta 8 hijos.

En el caso del *quadtree*, el dominio de entrada se describe utilizando una polilínea que delimita su contorno. Las técnicas de malla basadas en *quadtree* generan una malla *2d* a partir

de un cuadrante, que es un cuadrado que encapsula la polilínea de entrada. Este cuadrante se dividirá recursivamente en cuatro nuevos cuadrantes (cuadrados) hasta que se produzca la malla objetivo. En el contexto de una malla 3D, el dominio de entrada es una malla de superficie (o una malla tridimensional hueca). Este proceso opera de manera análoga, pero utilizando una estructura basada en el *octree*. Se comienza dividiendo el octante inicial, un cubo que engloba la malla inicial, de manera recursiva en ocho nuevos octantes hasta alcanzar el resultado deseado.

En el caso del *quadtree*, una celda C puede subdividirse en cuatro nuevas celdas. En este sentido, podemos decir que C se ha refinado. De aquí nace el concepto de "nivel de refinamiento" de una celda o *Refinement Level* (*RL*) en inglés, el que corresponde a un entero que indica cuantas veces han sido divididos sus padres hasta poder llegar hasta él. Esto se aprecia en la Figura 1.1, donde se muestran los niveles de refinamiento que tienen cada una de las celdas, algunas de ellas están pasando por el proceso de refinamiento. En esta se puede notar que, entre más alto es el nivel de refinamiento que tiene una celda más fina o pequeña es.

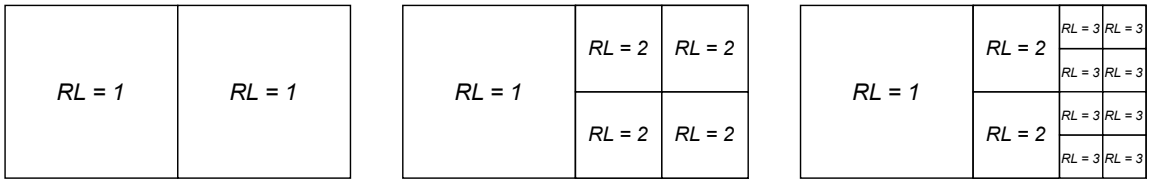


Figura 1.1: Ejemplo de nivel de refinamiento.

Debido a que este proceso de división no es uniforme, diferentes celdas pueden tener diferentes niveles de refinamiento, lo que significa que la malla resultante no será congruente. Además, en este punto queda claro que la única malla congruente posible será aquella en la que todas las celdas estén refinadas al mismo nivel, que es una malla tipo grilla. Para evitar esto, existe un conjunto de patrones de transición que se pueden usar cuando el *RL* entre cuadrantes vecinos no es mayor que uno [11].

Los patrones de transición definidos en [6], pueden ser aplicados tanto en mallas 2D como en

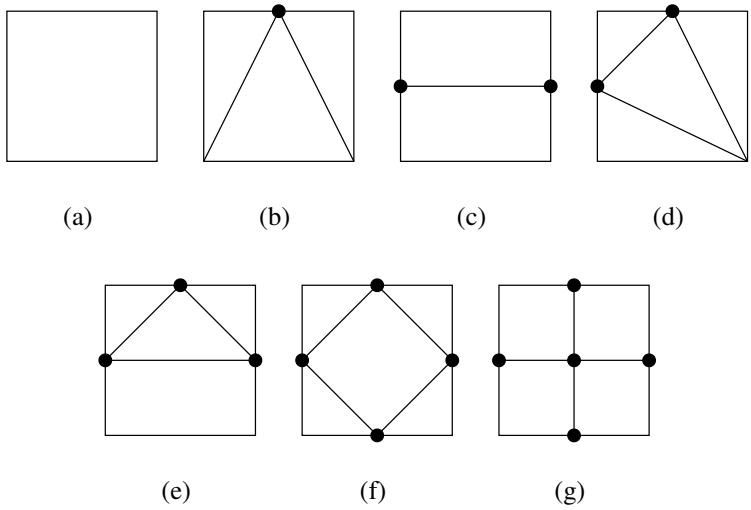


Figura 1.2: Patrones de transición para mallas 2D.

3D. Para el caso 2D, la cantidad de patrones de transición existentes es 7. En la Figura 1.2 se muestran todos los patrones de transición para las mallas de dos dimensiones. Por otro lado, para el caso de las mallas 3D, el número de patrones posibles asciende a un total de 325. En la Figura 1.3 se puede apreciar un par de ejemplos de patrón de transición para una malla 3D.

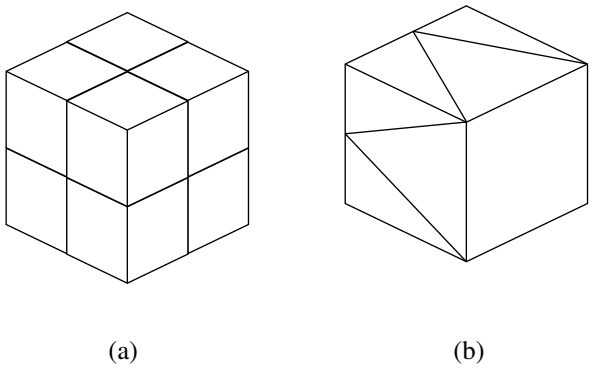


Figura 1.3: Ejemplo Patrones de transición para mallas 3D.

Un *quadtree* u *octree* que cumpla con la regla que el *RL* entre celdas vecinas no es mayor que uno se dice que está balanceado. El ejemplo que se muestra en Figura 1.4a es un *quadtree* que está balanceado y la Figura 1.4b muestra una malla congruente construida a partir del *quadtree* cuando se utilizan los patrones de transición.

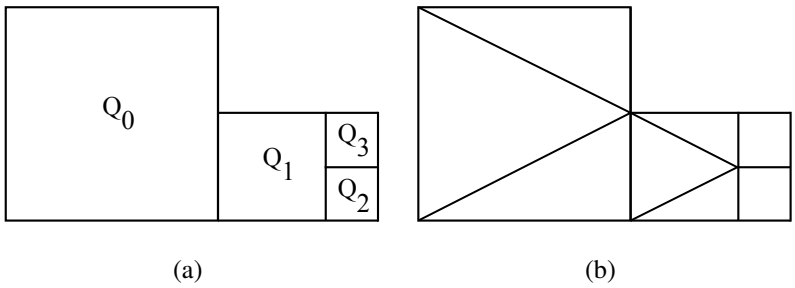


Figura 1.4: Ejemplo de cuadrantes vecinos.

Debido a esto, el principal objetivo es producir una malla balanceada para poder utilizar estos patrones de transición y finalmente poder producir una malla congruente. Es decir, cada vez que dos celdas vecinas tengan una diferencia en sus *RL* mayor que uno, el de menor nivel, es decir, la celda de mayor tamaño, debe ser refinado (dividido en cuatro nuevos cuadrantes u ocho nuevos octantes), por lo tanto, conocer el *RL* de los vecinos se convierte en una tarea crítica.

En este proceso de balanceo podría producirse una propagación del refinamiento. Esto ocurre porque cada vez que se refina una celda, incluso si se activa mediante balanceo, se debe verificar que sus vecinos también cumplan con esta regla de forma recursiva. Centrándose en el cuadrante Q_3 de la Figura 1.4a, podemos notar que este tiene un $RL = 3$, mientras sus que sus vecinos a los cuadrantes Q_1 y Q_2 tienen $RL = 2$ y $RL = 3$ respectivamente. Si Q_3 se refina, Q_2 seguirá estando equilibrado; sin embargo, eso no será así para Q_1, como se muestra en la Figura 1.5a. Por lo tanto, se tendría que refinar Q_1, que es el único cuadrante desbalanceado en ese momento. Al refinar Q_1, como se muestra en la Figura 1.5b, sus vecinos derechos se considerarán balanceados; sin embargo, Q_0 quien tiene un $RL = 1$, ahora se ha desbalanceado, por lo que debe refinarse y, una vez más, sus vecinos estarán balanceados, como se aprecia en la Figura 1.5c. Sólo en este momento toda la malla habrá recuperado el balance. Cabe destacar que cada vez que se refina una celda, se debe actualizar la información de sus vecinos respecto a las nuevas celdas generadas.

En el presente trabajo de tesis se propone una nueva estructura de datos basada en los nodos de la malla, con el objetivo de optimizar el proceso de balanceo en mallas de tipo *quad-tree*/*octree*. La estructura de esta tesis se distribuye de la siguiente manera: en el capítulo 2

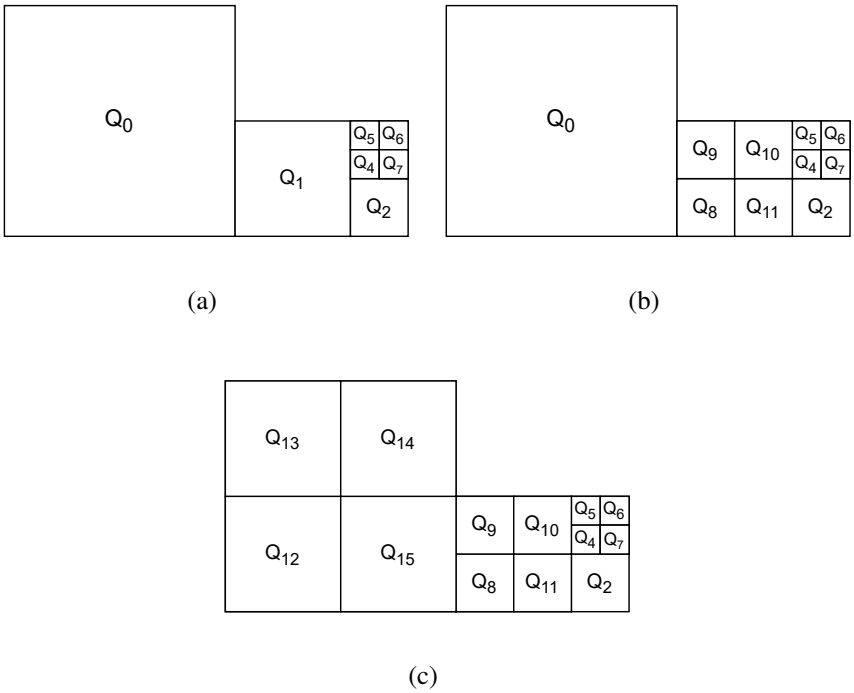


Figura 1.5: Ejemplo refinamiento vecinos.

se hará una revisión de la literatura respecto a las propuestas actuales que permiten almacenar la información de las celdas y sus vecinos. El capítulo 3 introduce la problemática que se abordará y plantea la hipótesis que se quiere probar. El capítulo 4 se detalla la propuesta de la estructura de datos de nodos y su funcionamiento. El capítulo 5 muestra los experimentos realizados y sus resultados para mallas en 2D y 3D. Finalmente, en el capítulo 6 estarán las conclusiones, se analizarán los resultados obtenidos y se discutirá el cumplimiento de los objetivos por parte de la propuesta realizada en este trabajo.

Capítulo 2

Estado del Arte

Debido a la complejidad de mantener en memoria la información de cada celda y sus vecinos, surge la necesidad de crear una estructura de datos que permita consultar esta información y actualizar estos datos durante el proceso de refinamiento. A continuación, se presentan algunas de las principales aproximaciones actuales que buscan resolver el problema de almacenar y actualizar dinámicamente los datos en una malla geométrica de *quadtree/octree*.

2.1. Estructuras de Datos Basadas en Árboles

Las estructuras de datos basadas en árboles son una de las alternativas más intuitivas y populares en el estado del arte. Estos árboles representan la "jerarquía" entre las celdas y las subceldas generados como resultado del proceso de refinamiento. En trabajos como [12], se busca generar o representar un árbol, donde cada nodo no terminal del árbol debe tener cuatro o ocho hijos dependiendo si la malla es 2D o 3D. Para realizar la tarea de encontrar las celdas vecinas, se debe comenzar desde un nodo terminal del árbol y navegar a lo largo del árbol. Este proceso se puede lograr de varias maneras, por ejemplo, como se muestra en [13] o [14].

Desafortunadamente, la necesidad de navegar por el árbol para encontrar un vecino tiende a tener un rendimiento deficiente al aumentar el número de celdas, debido a que: 1) lleva

tiempo encontrar el vecino siguiendo la estructura de datos del árbol y 2) cada celda que ya se ha refinado solo gasta más memoria, pero nunca será la respuesta a una consulta de vecinos, solo uno de sus hijos podría serlo. Es por eso que el objetivo de varios trabajos es hacer que el proceso de actualización de la información del árbol sea más eficiente, utilizando algún tipo de mejora durante las operaciones de consulta del árbol.

Por otro lado, el trabajo en [15] afirma que es posible, al menos teóricamente, obtener en tiempo constante el cuadrante vecino utilizando un *quadtree* lineal, que es una representación en forma de *Array* del árbol que solo contiene las hojas del mismo. Su trabajo se realiza en el contexto del procesamiento de imágenes y aquí, los cuadrantes se refinan según el nivel de precisión que se requiere. Para identificar los cuadrantes vecinos, utilizan un sistema de asignación de códigos únicos para cada cuadrante y luego, con el uso de operaciones binarias, pueden recuperar la ubicación del vecino. De esta manera, logran acceder a los datos de los cuadrantes vecinos en tiempo constante, asumiendo que una posición del *Array* se puede acceder en tiempo constante.

Este trabajo se continúa más adelante en [16] con las pruebas experimentales de esta estructura de datos, obteniendo mejores resultados que los árboles tradicionales, tanto para imágenes sintéticas como reales. Sin embargo, este trabajo no se centra en un *quadtree* equilibrado, por lo que no está claro si este resultado podría usarse cuando el refinamiento de un cuadrante puede afectar el refinamiento de un vecino. Además, en esta propuesta, no es una tarea importante identificar el nodo intermedio de un borde, porque esta tarea no es relevante en el procesamiento de imágenes, mientras que en las mallas que se estudian en este trabajo es algo crítico ya que, dependiendo de la cantidad y disposición de estos puntos, se debe seleccionar el patrón de transición adecuado. Por esta razón, no es obvio si este enfoque se puede usar de manera eficiente durante el proceso de balanceo. Finalmente, esta estrategia se contextualiza en 2D, por lo que, a diferencia de propuestas anteriores, adaptar este trabajo a otros tipos de mallas, como el tipo *octree* en 3D, no está claro que sería una tarea sencilla.

2.2. Estructuras de Datos Basadas en Arcos

En [10] se presenta una propuesta de algoritmo de mallado con elementos mixtos, utilizando un enfoque que trata el balanceo de cuadrantes y octantes mientras se ejecuta el proceso de refinamiento, obteniendo mejores resultados que tratar el balanceo una vez finalizado todo el proceso de refinamiento. En este algoritmo, se presenta una estructura de datos basada en los arcos de la malla para el proceso de mantener el balance.

Para el caso de las mallas 2D, esta estructura de datos consta de un *Array* de tres espacios como se muestra en la Tabla 2.1. El primero es el índice del nodo medio, en el caso que lo hubiera, y los dos espacios siguientes son para los índices de los dos cuadrantes que comparten esa arista. Se debe tener en cuenta que uno de ellos podría estar vacío, en el caso de los cuadrantes en el límite del dominio.

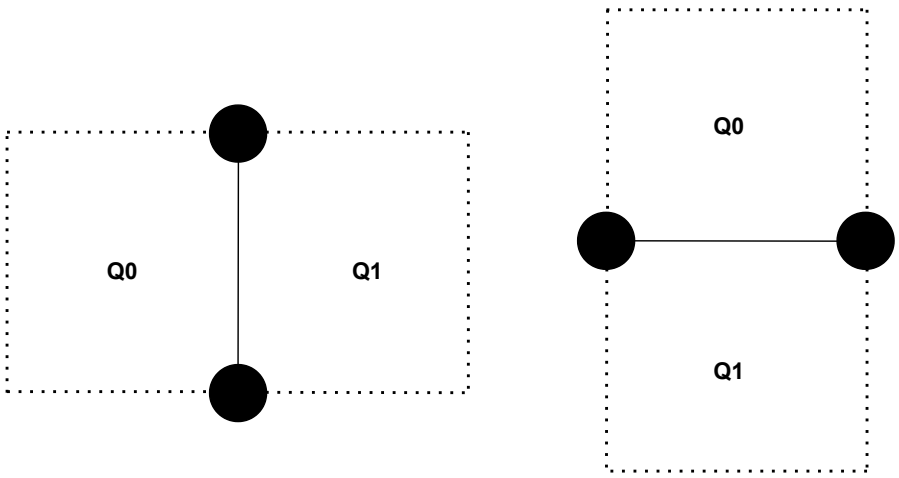
Tabla 2.1: Estructura de datos de arcos en 2D

Array de enteros sin signo		
Nodo medio	Cuadrante vecino Superior o Izquierdo	Cuadrante vecino Inferior o Derecho

La orientación de los cuadrantes vecinos que ocuparan los dos espacios en el *Array* de la estructura de arcos dependerán de la orientación de dicho arco. Como se muestra en la Figura 2.1a, en el caso de un arco vertical, se almacenará la información de los cuadrantes izquierdo y derecho. Por otro lado, para el caso del arco horizontal, como se muestra en la Figura 2.1b, esta estructura almacenará la información de los cuadrantes superior e inferior.

Por otro lado, para el caso de las mallas de dominio en 3D, esta estructura de datos consta de un *Array* de cinco espacios como se muestra en la Tabla 2.2. El primero es el índice del nodo medio, en el caso que lo hubiera, y los cuatro espacios siguientes son para los índices de los cuatro posibles octantes que comparten esa arista. Se debe tener en cuenta que más de uno de ellos podría estar vacío, en el caso de los cuadrantes en el límite del dominio.

Al igual que el caso en 2D, los octantes vecinos de los arcos dependerá de la orientación del



(a) Arco vertical en malla 2D. (b) Arco horizontal en Malla 3D.

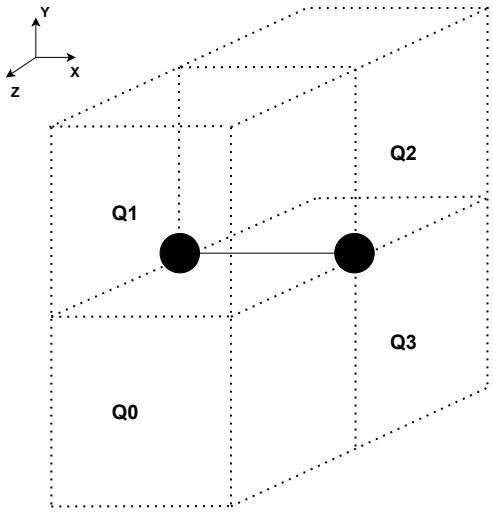
Figura 2.1: Arco en malla 2D

Tabla 2.2: Estructura de datos de arcos en 3D

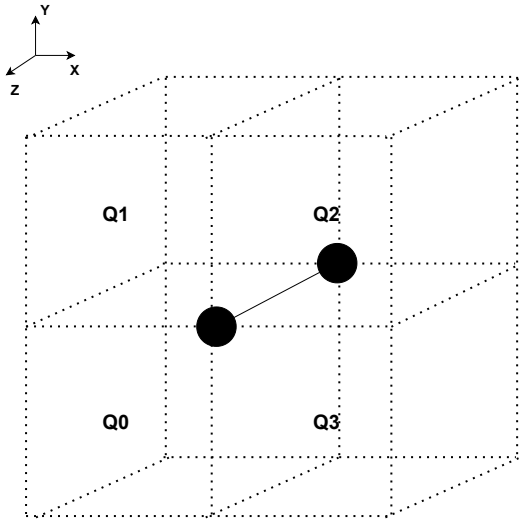
Array de enteros sin signo				
Nodo medio	Primer octante vecino	Segundo octante vecino	Tercer octante vecino	Cuarto octante vecino

arco. En la Figura 2.2a se muestra los octantes vecinos que almacenará un arco en el eje x, que serían los octantes en los ejes Y y Z que comparten dicho arco. Por otro lado, en las Figuras 2.2b y 2.2c se muestran arcos en los ejes Z e Y respectivamente, donde también se evidencia los octantes vecinos en la malla 3D.

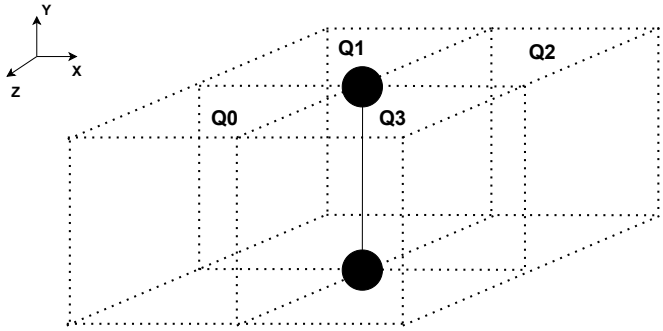
Cabe mencionar que estos *Array* que almacenan la información de los arcos son almacenados en un *Map* de la *STL* de C++, donde los índices de los nodos que componen el arco son la llave y el valor es el *Array* con la información anteriormente mencionada. Un problema con esta alternativa es el tiempo que toma agregar más arcos al *Map* y buscar arcos dentro de esta estructura. Esta operación es relevante porque durante el proceso de refinamiento de la malla, cuando se debe realizar una validación del balance entre los vecinos, cada arco de las celdas debe ser verificado, lo que significa que esta operación se utilizará muchas veces.



(a) Arco eje X en malla 3D.



(b) Arco eje Z en malla 3D.



(c) Arco eje Y en Malla 3D.

Figura 2.2: Arco en malla 3D

A pesar de este problema, la estructura de datos de arcos funciona mejor que las estructuras de datos basadas en árboles. Esto se debe a que, por un lado, para obtener el vecino de una celda durante la verificación del balanceo, basta con buscar en el *Map* el arco que se está comprobando y este contiene la información de los vecinos de la celda. Por otro lado, las estructuras basadas en árboles deben navegar a lo largo del árbol para encontrar a su vecino. Además, durante el proceso de balanceo, se debe verificar si previamente insertaron un nodo medio en los arcos para evitar la duplicación de nodos en la malla. Dado que esta información se almacena en el *Array* de los arcos, se accede directamente a ella.

Otro trabajo que utiliza el enfoque basado en arcos, es [17], donde se presenta un algoritmo para generar un nuevo tipo de malla poligonal obtenida a partir de triangulaciones. Estos polígonos se construyen a partir de una región de arcos que sean terminales, vale decir, los del borde de la malla, donde se van reduciendo los arcos que no sean el más largo de los dos triángulos que lo comparten. Este algoritmo denominado *Polylla*, genera una malla de polígonos con forma convexa y no convexa. En este trabajo se comparan en desempeño con las mallas basadas en Voronoi y logran concluir que el rendimiento de las mallas *Polylla* y mallas de *Voronoi* es similar. Sin embargo, como esta propuesta utiliza mallas no basadas en el *quadtree/octree* no es un punto de comparación para este trabajo.

2.3. Estructuras de Datos Basadas en Nodos

En [18], se presenta la estructura de datos *Start-Vertex* como una opción para representar mallas planas generales, permitiendo un tiempo constante de consulta de adyacencia y un menor uso de memoria en circunstancias específicas. Esta estructura de datos guarda la información de conectividad de los nodos de tal manera que es posible conocer el siguiente nodo en una cara particular de la malla sin necesidad de guardar la cara en sí. Un ejemplo de una estructura de *Start-Vertex* se representa en la Figura 2.3, con las flechas de los vértices señalando hacia todos los vecinos posibles de los nodos en una cara de la malla. Cabe notar que, a diferencia de los otros nodos, v_0 pertenece solo a esta cara pentagonal.

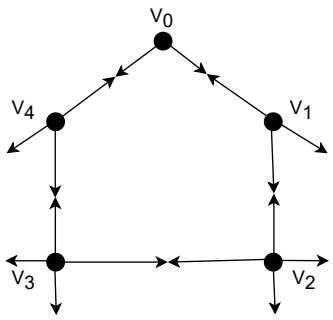


Figura 2.3: Ejemplo de *Start-Vertex* en la cara triangular de una malla.

Dado que esta propuesta se centra en hacer compacta la representación de las caras de las

mallas planas, no está claro si esta estructura podría utilizarse durante el proceso de refinamiento en la verificación del balance entre cuadrantes. Probablemente necesitaría alguna adaptación para identificar al menos los cuatro nodos originales del cuadrante.

Capítulo 3

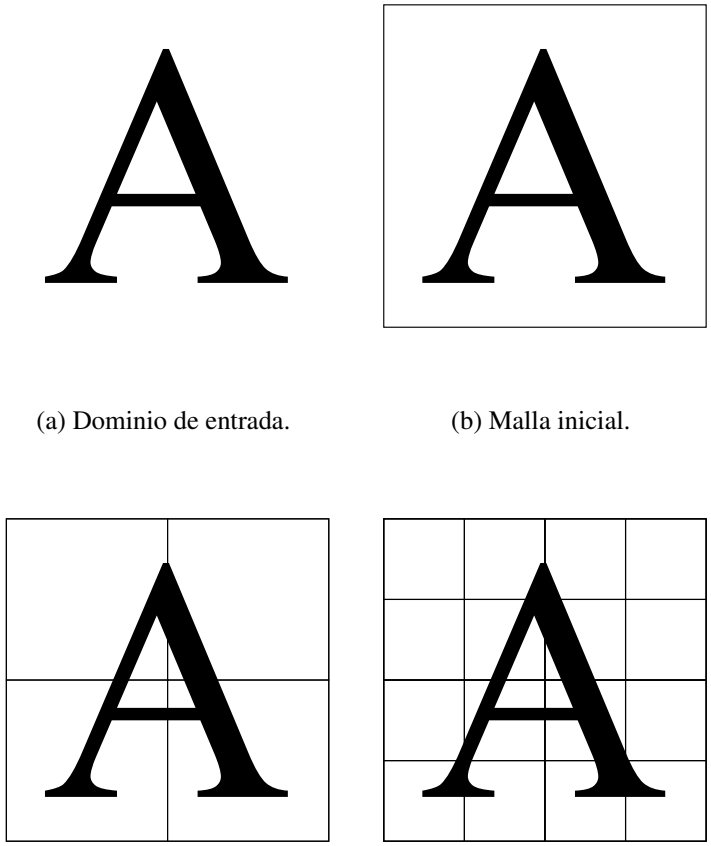
Hipótesis

3.1. Generación de Mallas basadas en Quadtree y Octree

El algoritmo generador de mallas basadas en quadtree/octree presentado en [10], busca crear representaciones de los objetos representados por el dominio de entrada, ya sea una polilínea o una malla de superficie. Tanto para la generación de mallas de superficie o de volumen, el dominio de entrada se encierra en una celda inicial, que se irá diviendo tantas veces como indique el nivel de refinamiento que se desea alcanzar.

En la Figura 3.1a se muestra el dominio de entrada para la generación de una malla geométrica. Este dominio es encapsulado en un cuadrante inicial, como se aprecia en 3.1b, para luego comenzar a ser refinado hasta el nivel de refinamiento deseado, lo que va generando progresivamente un *quadtree* más fino como se muestra en el cambio de la Figura 3.1c a la Figura 3.1d.

Una característica adicional de este algoritmo, es que permite un refinamiento más alto en zonas que pueden ser determinadas por algún interés en particular. En la Figura 3.2a se muestra un *quadtree* balanceado con la región superior del dominio de entrada más refinado debido a que esta podría ser de interés para alguna simulación. Este algoritmo, además de ser capaz de refinar el *quadtree* y permitir seleccionar alguna zona de interés, debe eliminar los

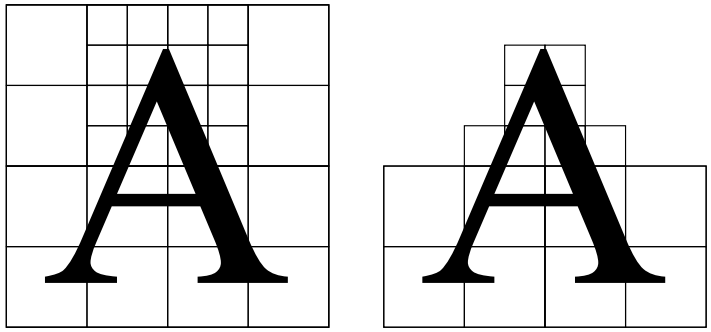


(a) Dominio de entrada. (b) Malla inicial.

(c) Primer nivel de refinamiento. (d) Segundo nivel de refinamiento.

Figura 3.1: Refinamiento de dominio de entrada.

cuadrantes que no intersectan con el dominio de entrada, es decir, los cuadrantes externos, permitiendo formar una malla más representativa al dominio de entrada. En la Figura 3.2b se muestra la malla resultante de la eliminación de cuadrantes que no contienen o intersectan con alguna parte del dominio de entrada.



(a) Refinamiento región superior. (b) Eliminación de cuadrantes.

Figura 3.2: Refinamiento región superior y eliminación de cuadrantes.

3.2. Conjetura

A raíz de la descripción anterior del algoritmo, y recordando que éste utiliza una estructura basada en arcos para almacenar la información, donde dichos arcos se almacenan en un *map*, se da paso a la conjetura para establecer la hipótesis de este trabajo.

Cuando una celda se divide en cuatro u ocho subceldas dependiendo de la dimensión de la malla, la celda original, que tiene un identificador único, deja de ser relevante para la malla ya que no se puede volver a refinar, en otras palabras, la celda fue reemplazada por sus subceldas en la malla. Por otro lado, las celdas que no intersectan con el dominio de entrada tampoco serán requeridas durante el resto del proceso de generación ya que se eliminan, como se ve en la diferencia entre las Figuras 3.2a y 3.2b. Es claro que almacenar información de celdas eliminadas, no es algo relevante para la creación de la malla.

Sumado a esto, durante el proceso de refinación de las celdas, se van generando más arcos que nodos de la malla tanto en 2D como en 3D. Como se muestra en la Figura 3.3, al ir refinando la malla para el dominio de entrada expuesto en la Figura 3.1a, la cantidad de arcos es siempre mayor a la cantidad de nodos generados.

En base a lo anterior, el enfoque de almacenar la información de la vecindad de las celdas en los arcos, es un proceso costoso debido a:

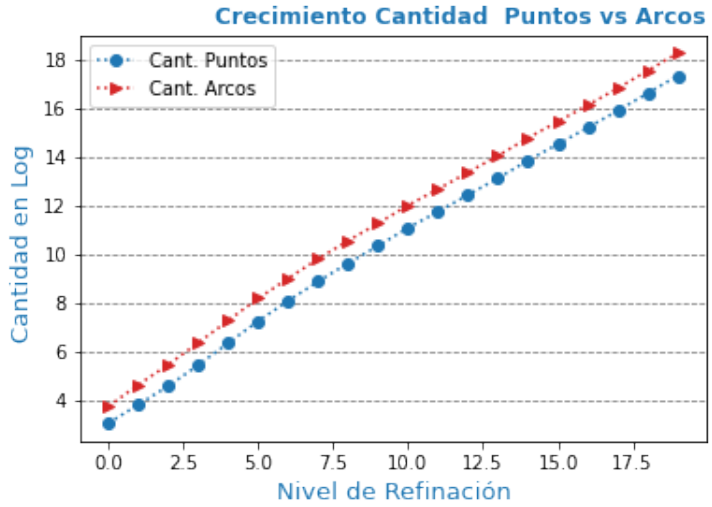


Figura 3.3: Cantidad de puntos y arcos presentes en la malla.

- El hecho de que hay más arcos que nodos en la malla a medida que se va refinando.
- Se almacena información de arcos por los que no se volverá a consultar, como es el caso de los arcos que pertenecen a celdas que ya se refinaron junto con sus vecinos.
- El uso de un *Map* para almacenar los arcos es costoso respecto al tiempo que toman las operaciones de consultas sobre él, junto con el costo en memoria.

a que esto guarda información de arcos por los que no se volverá a consultar, como sucede con las celdas ya refinadas. Por otro lado, debido al uso de los *map* para almacenarlos es un proceso que podría ser muy costoso tanto en tiempo o en memoria, dependiendo las características del dominio de entrada. Además, como la cantidad de arcos presentes en la malla es mayor a la cantidad de nodos, el cambiar el enfoque de almacenamiento de información a los nodos favorecería a una generación más rápida de la malla geométrica.

3.3. Hipótesis de Trabajo

En base a lo anterior, se plantea la siguiente hipótesis de trabajo:

Mediante una estructura de datos asociada a los nodos de una malla Quadtree/Octree, es posible reducir el tiempo de búsqueda de las celdas vecinas y la obtención de nodos intermedios de un arco en una celda de $O(\log n)$ *a* $O(1)$

3.4. Objetivos

3.4.1. Objetivo Generales

- Definir una estructura de datos basada en los nodos de la malla geométrica tipo *quadtree/octree*, que permita obtener información respecto a sus celdas vecinas e identificar el nodo medio de un arco en tiempo $O(1)$.
- Disminuir el tiempo global de la creación de mallas geométricas basadas en *quadtree/octree*.

3.4.2. Objetivo Específicos

- Diseñar una estructura de datos con información local de los nodos de un *quadtree/octree* que permita identificar a las celdas vecinas en tiempo $O(1)$.
- Diseñar una estructura de datos con información local de los nodos de un *quadtree/octree* que permita identificar la presencia de puntos intermedios en un arco de una celda en tiempo $O(1)$.
- Implementar la estructura de datos basada en nodos propuesta, para concluir si es una solución competitiva, en términos de tiempo de ejecución y memoria utilizada, para el problema planteado.

Capítulo 4

Propuesta

Dada las alternativas existentes en el estado del arte, y recordando que la tarea principal es optimizar el acceso a la información de las celdas vecinas, en lugar de utilizar un enfoque basado en almacenar la información en las celdas, como es en el caso de las estructuras de datos basadas en árboles, o almacenar la información en los arcos de las celdas, se propone la alternativa de almacenar los datos en los nodos de la malla.

La motivación detrás de este nuevo enfoque radica en que con las propuestas actuales, tanto las que usan un enfoque basado en arboles como el basado en arcos, se almacenan datos que en la malla no se volverán a necesitar, como lo son celdas que no existen por que ya fueron refinadas o arcos por los que no se volverá a preguntar debido a que ya sufrieron todos los procesos de refinamiento que admiten. Por otro lado, con esta propuesta, los nodos que existan en la malla siempre podrían ser requeridos en la refinación de algún posible subcuadrante o suboctante, siempre y cuando permanezca en la malla al menos una de las celdas de las que dicho nodo pertenece.

4.1. Estructura de Datos Basada en Nodos

Cada cuadrante y cada nodo de la malla tendrá un identificador único tanto para las mallas en dos y tres dimensiones. En el caso de las mallas *quadtree* en 2D, cada nodo almacenará la

siguiente información: nodos vecinos directos, cuadrantes vecinos y sus respectivos *RL*. Los datos se almacenan en tres grupos de cuatro enteros sin signo cada uno, como se muestra en la Tabla 4.1. El primer grupo corresponde a los índices de los nodos vecinos en cada una de las cuatro direcciones posibles. El segundo grupo corresponde a los índices de los cuatro posibles cuadrantes de los que podría formar parte el nodo. Y finalmente el último grupo corresponde al *RL* de los cuadrantes del grupo anterior respectivamente.

Tabla 4.1: Estructura de datos de nodos en 2D

***Array* de enteros sin signo**		
Cuatro nodos vecinos	Cuatro cuadrantes vecinos	Cuatro niveles de refinamiento

Para el caso en 3D, cada nodo almacenará la siguiente información: nodos vecinos directos, octantes vecinos y un código identificador en cada eje. Los datos se almacenan en tres grupos, dos de seis enteros sin signo y uno de ocho enteros sin signos, como se muestra en la Tabla 4.2. El primer grupo corresponde a los índices de los nodos vecinos en cada una de las seis posibles direcciones. El segundo grupo corresponde a los índices de los ocho posibles octantes de los que podría formar parte el nodo. El último grupo corresponde a un código identificador como punto medio del arco que lo contiene en cada uno de los tres ejes posibles. Es una tupla por cada eje, donde los valores en cada tupla son los identificadores, ordenados de mayor a menor, de los nodos del arco que lo contienen, donde el nodo en cuestión es el punto medio. Este mecanismo se detallará más adelante, junto con un ejemplo.

Tabla 4.2: Estructura de datos de nodos en 3D

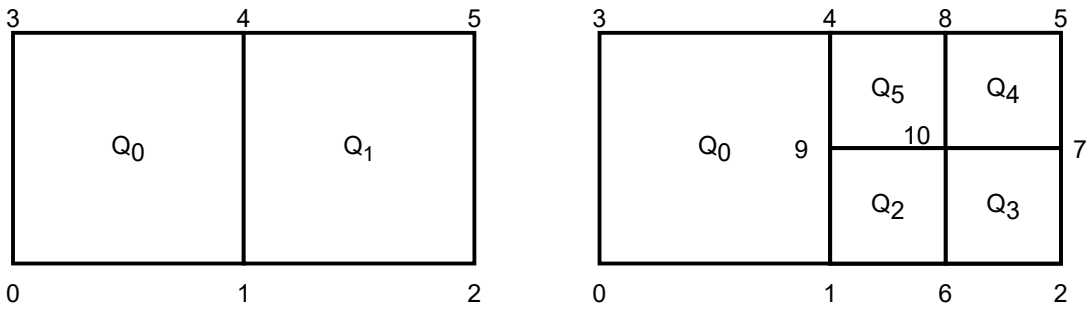
***Array* de enteros sin signo**		
Seis nodos vecinos	Ocho octantes vecinos	Código ejes *x y z*

Por cada nodo de la malla, se creará un *array*, con las características antes mencionadas, en un vector donde la posición en el vector será el identificador del nodo. De esta forma,

para acceder a la información almacenada en un nodo, basta con acceder directamente al vector utilizando el identificador del nodo, evitando por completo el uso de punteros, lo que predomina en las propuestas actuales. Este criterio se aplicará tanto en el caso en 2D y en 3D. Esta característica permite que esta propuesta sea mucho más eficiente en el proceso de identificar a las celdas vecinas y comprobar si están equilibrados, ya que, suponiendo que el tiempo para acceder a una posición de un vector es constante, entonces se podría comprobar si los vecinos de un cuadrante u octante están equilibrados en tiempo constante.

4.2. Proceso de Actualización de Datos

Durante el proceso de refinamiento de una celda, se debe actualizar la información del nodo almacenada en su *array*. En este proceso, cada nodo actualizará sus nuevos nodos vecinos, nuevos cuadrantes u octantes vecinos y deberá incrementar los *RL* respectivos en caso 2D. Estos cambios sólo se harán en la dirección correspondiente a la posición del nodo en la celda, por lo que no se debe modificar toda su información, sólo la que está relacionada con los nodos, cuadrantes u octantes y niveles que se ven afectados por la celda que se está refinando.



(a) Cuadrante Q_1 sin refinar.

(b) Cuadrante Q_1 refinado.

Figura 4.1: Proceso refinamiento Q_1.

En la Figura 4.1 se muestran algunas de las etapas del proceso de refinamiento del cuadrante Q_1 en una malla en 2D. Considerando los datos almacenados por el nodo 1, en la Figura 4.1a, este nodo tiene registrados los nodos 0, 2 y 4 como vecinos, y Q_0 y Q_1 como cuadrantes

vecinos, ambos con nivel de refinamiento 1. Luego, al dividir el arco 1-2 en la Figura 4.1b, el nodo 1 debe actualizar que su nodo vecino derecho ahora es el nuevo nodo 6, junto con su nuevo cuadrante vecino superior derecho que es Q_2 con nivel de refinamiento 2. Cuando se crea el último subcuadrante Q_5 en la Figura 4.1b, dado que este cuadrante usa el nuevo nodo 9 que está en el medio del antiguo arco 1-4, el nodo 1 deberá actualizarse para indicar que su nuevo vecino superior es el nodo 9.

4.3. Identificación de Nodo Medio de un Arco

Durante el proceso de refinamiento de la malla, podría ocurrir un desbalance entre dos celdas vecinas, por lo que la de menor nivel de refinamiento deberá refinarse. En este proceso, si sus cuadrantes u octantes vecinos ya se refinaron, estos ya agregaron nodos medios en los arcos que comparten, por lo tanto, este proceso de refinación debe detectar que estos nodos ya existen, identificarlos y utilizarlos como parte del refinamiento en vez de crear otros nuevos. Esta estructura propuesta soporta esta característica tanto para el caso en 2D y 3D, sin embargo, funciona de forma diferente dependiendo de la dimensión.

4.3.1. Caso Malla 2D

Como se muestra en la Figura 4.2a, Q_1 tiene un $RL = 1$, mientras que su cuadrante vecino Q_8 tiene un nivel 3. Por lo tanto, Q_1 debe refinarse para aumentar su RL en uno y cumplir con la regla del balanceo. Al iniciar este proceso, el algoritmo debe darse cuenta que el nodo intermedio ya existe en el arco 1-4, debido a que sabe que está desbalanceado, por lo tanto, ahora debe identificar cuál es el nodo intermedio.

En las figuras 4.2a y 4.2b hay dos nodos en el arco 1-4, pero su orden está invertido. En ambos casos, el nodo medio es el nodo 7 y debe usarse en el refinamiento de Q_1. Para identificar este nodo, el algoritmo debe revisar todos los nodos presentes en los arcos 1-4 y verificar sus niveles de refinamiento en la dirección de la celda vecina ya refinada. En ambas figuras, el nodo 12, que no es el nodo intermedio, tiene almacenado que los niveles de refinamiento de

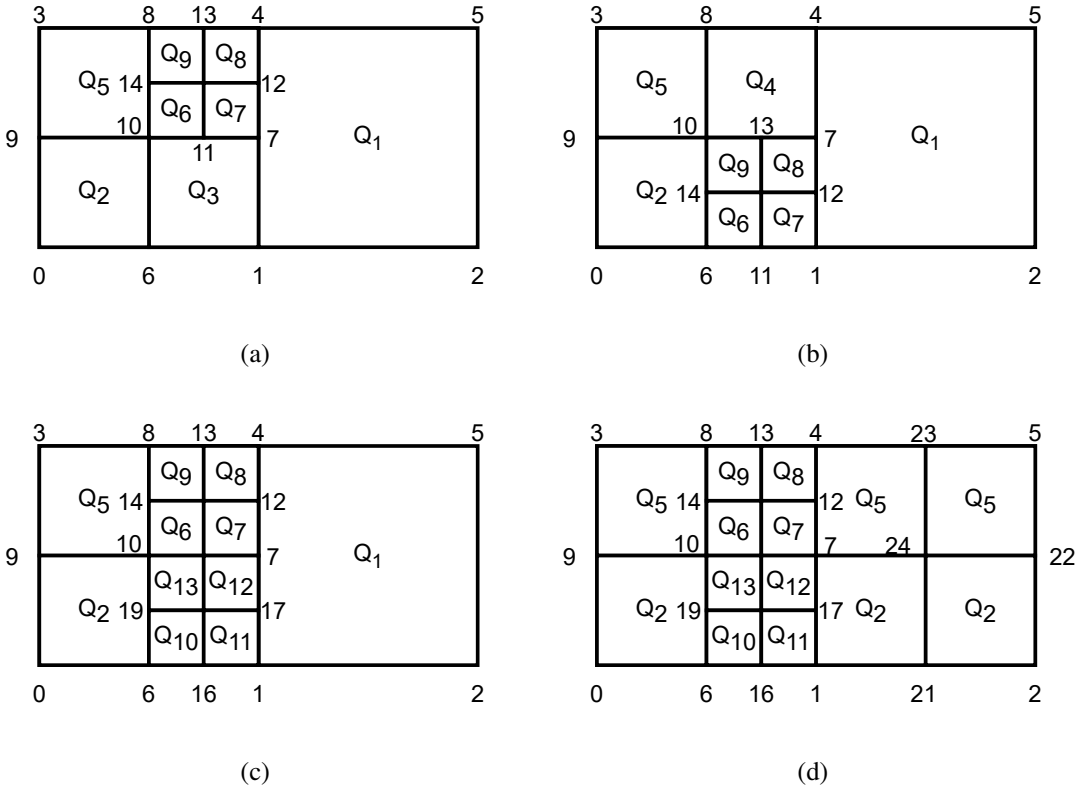


Figura 4.2: Proceso refinamiento Q_1 desbalanceado.

los cuadrantes a su izquierda son los mismos, ya que tanto Q_8 como Q_7 tienen el nivel de refinamiento 3. Por otro lado, el nodo 7 tiene que sus niveles de refinamiento son diferentes. En la Figura 4.2a sus vecinos izquierdos son Q_3 y Q_7, y en la Figura 4.2b sus vecinos izquierdos son Q_4 y Q_8. En ambos casos uno de ellos tiene nivel de refinamiento 2 y el otro 3, por lo tanto, identificar este nodo equivaldría a ver qué nodo tiene cuadrantes vecinos con diferente nivel de refinamiento.

Sin embargo, lo anterior no es suficiente para el caso de que los dos cuadrantes vecinos izquierdos de Q1 ya hayan sido refinados. Como se muestra en la Figura 4.2c, este es la última combinación que puede ocurrir y todos los nodos del arco 1 a 4 tienen el mismo nivel de refinamiento. Para esta situación, bastaría con comprobar los nodos y elegir el del medio. Cabe señalar que, debido a la regla de balanceo, al ejecutarse justo cuando se ha perdido el equilibrio entre dos cuadrantes vecinos, nunca puede haber más de tres nodos insertados en una arista. Es decir, entre dos cuadrantes que tienen *RL* diferente, la arista

que comparten puede tener de 0 a 3 nodos como máximo. Cabe mencionar que, durante el proceso de refinamiento de una malla *quadtree*, luego de refinar algún cuadrante y notar que alguno de sus vecinos está desbalanceados. Estos últimos serán agregados a una lista de cuadrantes desbalanceados que serán trabajados una vez se termine de refinar todos los cuadrantes de la presente iteración seleccionados según el tipo de refinamiento de la malla. En otras palabras, la Figura 4.2c es el resultado de que Q_3 y Q_4 están en proceso de ser refinados en esta iteración, notando ambos que Q_1 estaba desbalanceado, y este se refinó posterior a que terminara la lista anteriormente mencionada, dejando como resultado lo que se muestra en la Figura 4.2d.

4.3.2. Caso Malla 3D

Como se mencionó en la sección 4.1, para el caso en tres dimensiones la estructura de datos de nodos remplaza los datos respecto a los niveles de refinamiento por un código en forma de tupla irrepetible entre los nodos en cada eje. Este cambio se produjo debido a la complejidad a la que escala utilizar la técnica mencionada anteriormente en tres dimensiones, debido a que se debe segmentar dependiendo la orientación del arco para verificar los niveles de refinamiento correspondientes.

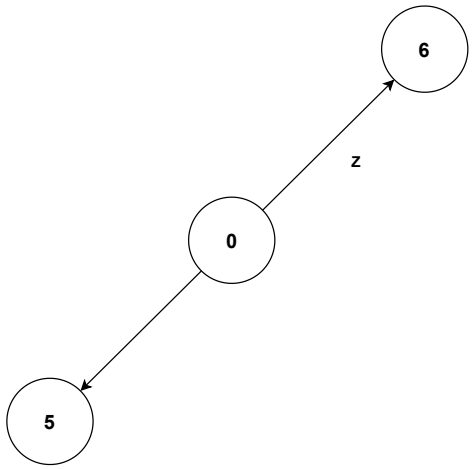


Figura 4.3: Arco de malla volumétrica en eje Z.

Debido a que los nodos poseen identificador únicos todos los arcos en la malla volumétrica

estarán compuestos por distintos pares de nodos y podrían tener orientación en uno de los tres ejes posibles. De esta manera, si se agrega un nodo en el medio de un arco, este nodo puede almacenar los valores de los nodos extremos de la arista junto con la orientación, y esta combinación sería única en toda la malla.

En la Figura 4.3 se muestra el arco 5-6 en el eje Z. En este arco se agregó un nodo medio, que por simplicidad del ejemplo, identificado como el nodo 0. Dado que este arco es único debido a que los identificadores de los nodos son irrepetibles, el nodo 0 puede almacenar en su información local, ordenando de mayor a menor, que los puntos extremos del arco del que es parte es la tupla $(6, 5)$. En base a esto, si se necesitará a futuro revisar cual es el nodo medio de dicho arco, bastaría recorrer el arco 5-6 y ver que nodo tiene almacenado en el eje Z el valor $(6, 5)$.

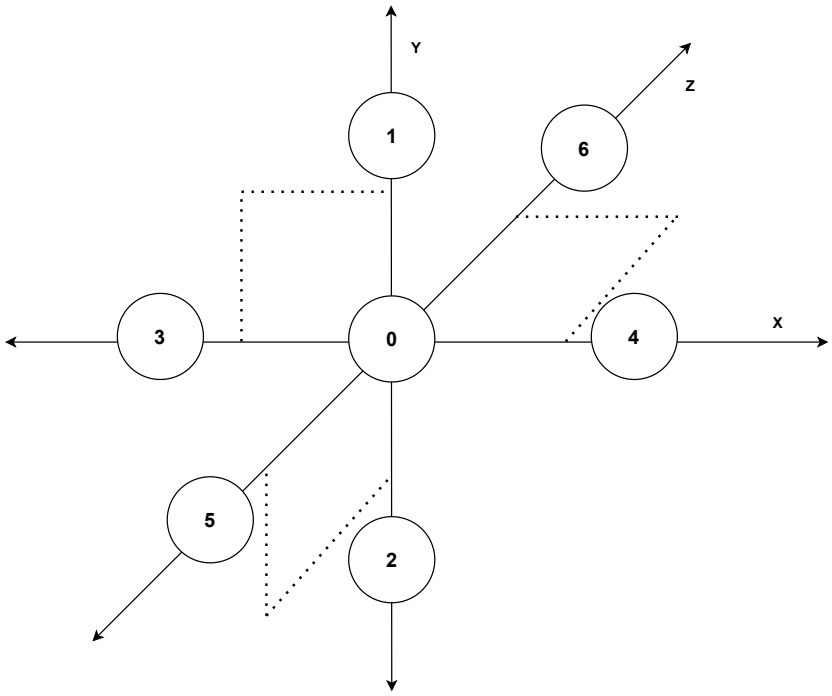


Figura 4.4: Arcos de malla volumétrica en ejes X, Y, Z.

El procedimiento anterior se puede repetir para almacenar los códigos correspondiente a los arcos de los que el nodo es parte en cada uno de los ejes posibles. En la Figura 4.4 se muestra que el nodo 0 es el nodo medio del arco 3-4 en el eje X, 1-2 en el eje Y, y del arco 5-6 en el eje Z. En base a esto, siguiendo el procedimiento anterior, la información que almacenará

el nodo 0 sería la tupla $(4, 3)$ para el eje X, $(2, 1)$ para el eje Y, y $(6, 5)$ para el eje Z. Dado estos valores, si alguno de los arcos que lo contienen necesita saber el nodo medio, bastaría revisar los nodos presentes en la arista y que el valor almacenado corresponda con la tupla formada por los identificadores de los nodos extremos.

En la Figura 4.5 se muestra el arco 3-4 en el eje X, cuyo nodo medio es el 0 pero los sub-arcos 3-0 y 0-4 ya han sido parte de un proceso de refinación, por lo poseen como nodo los nodos 7 y 8 respectivamente. En el caso de que algún octante desbalanceado tenga que refinarse y uno de sus arcos sea el 3-4, el necesitará obtener el nodo medio del arco. Como dicho octante sabe que los nodos 3 y 4 son los extremos, el algoritmo debe recorrer dicho arco y buscar el nodo que tenga como valor del eje X la tupla $(4, 3)$. Al recorrer esta arista, los valores que tendrán los nodos 7 y 8 en el eje X serán las tuplas $(3, 0)$ y $(4, 0)$ respectivamente. De esta forma, el algoritmo siempre detectará que el nodo 0 es el medio del arco ya que este tiene almacenado el valor $(4, 3)$. Cabe destacar, que de forma análoga a lo ocurrido en 2D, como todo octante que tenga una diferencia mayor a uno entre su nivel de refinamiento con el de alguno de sus vecinos, todas sus aristas podrían contener entre 0 a 3 nodos como máximo.

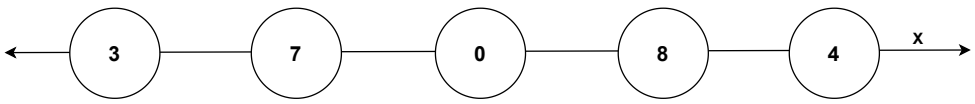


Figura 4.5: Arco de malla volumétrica en eje X.

Capítulo 5

Experimentos y Resultados

En esta sección, se presentarán algunos resultados representativos de varias ejecuciones que se han realizado tanto en mallas 2D como 3D. Para el caso de las mallas bidimensionales, se eligió una polilínea que representa la letra A que fue sometida a diversos tipos de refinamiento. Por otro lado, para el caso 3D, se eligió una malla de superficie representativa de la corteza cerebral humana. Para estos experimentos, se añadió la propuesta al algoritmo de mallado descrito en [10]. Estos resultados exhibirán las mejoras de esta estructura de datos, además de mostrar su eficiencia en términos de tiempo computacional y uso de memoria. También se realizarán algunas comparaciones con la estructura de datos de arcos. Ambas estructuras de datos, de arcos y de nodo, se escribieron en C++. Las pruebas para las mallas 2D se ejecutaron en un entorno Linux con una CPU Intel® Core™ i5-8600 a 3,10 GHz y 32 GB de RAM. Por otro lado, para las pruebes de mallas 3D fueron ejecutadas en un entorno Linux con una CPU Intel® Xeon® Gold 6230 a 2.10GHz y 256 GB de RAM. Las mediciones de tiempo están en milisegundos (ms) y las mediciones de memoria se redondean a megabytes (MB).

5.1. Resultados Mallas 2D

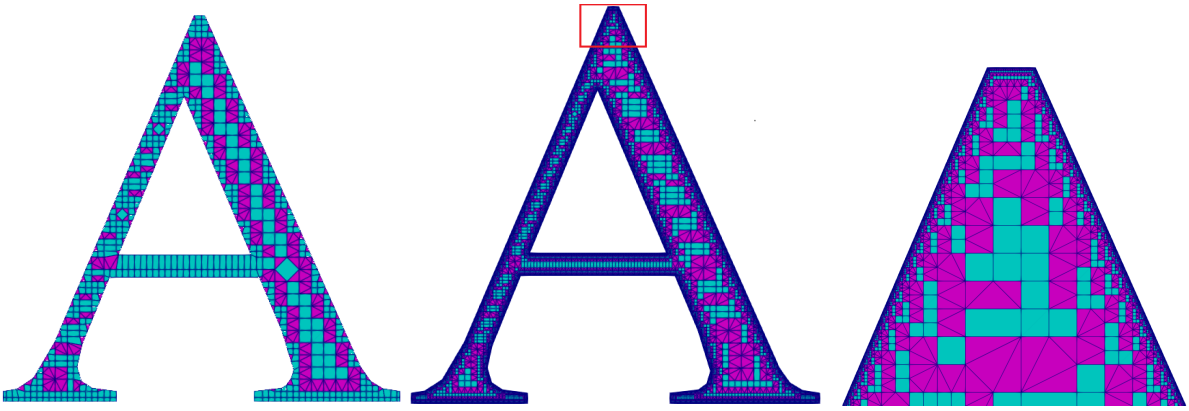
En es esta sección se detallarán los tipos de refinamiento a los que se someterá la polilínea que representa la letra A, junto con los resultados obtenidos de las diferentes mediciones de los tiempos de ejecución y el uso de memoria.

5.1.1. Tipos de Refinamiento

Los tipos de refinamiento que se realizarán para probar el desempeño de esta propuesta en mallas 2D son los siguientes:

Refinamiento de Borde

La prueba de refinamiento de borde consiste en refinar sólo los cuadrantes que intersectan con el borde de la polilínea de entrada, es decir, el contorno del dominio. Se contrastarán las estructuras de datos de nodos y arcos al ejecutar el algoritmo de mallado variando el parámetro *RL* del nivel seis al veinte. En la Figura 5.1 se muestran unos ejemplos de la mallas resultantes con estos parámetros.

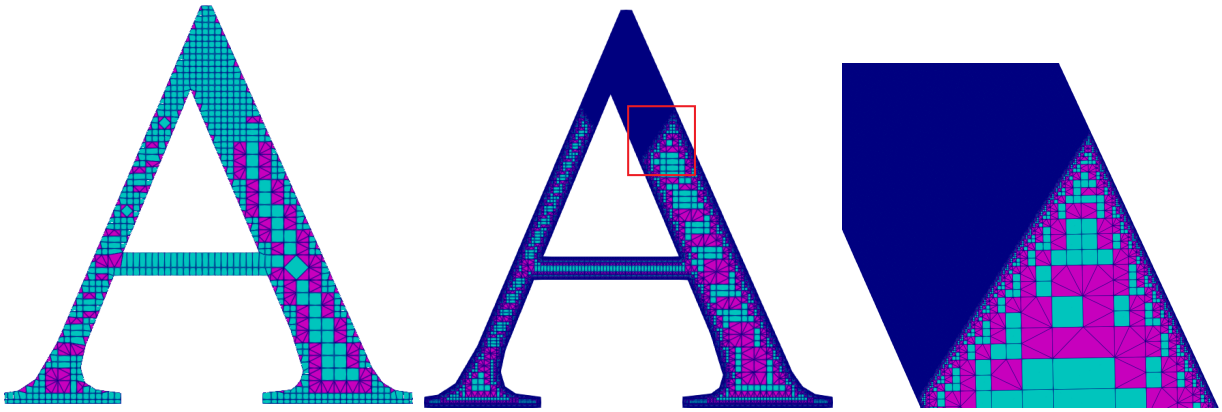


(a) Nivel de Refinamiento 6 (b) Nivel de Refinamiento 14 (c) Zoom refinamiento

Figura 5.1: Malla 2D resultante con refinamiento de borde.

Refinamiento de Región Superior y de Borde

En el algoritmo de mallado definido en [10], es posible definir una región particular donde se puede especificar un RL diferente. Esta región está definida por otra polilínea de entrada que debe intersectar el dominio. Entonces, en esta prueba la región superior y el borde de la malla se refinarán al mismo nivel. Debido al mayor uso de memoria por parte de estas pruebas, se midió hasta $RL = 14$. En la Figura 5.2 se muestran unos ejemplos de las mallas resultantes con estos parámetros.



(a) Nivel de Refinamiento 6 (b) Nivel de Refinamiento 14 (c) Zoom refinamiento

Figura 5.2: Malla 2D resultante con refinamiento de la región superior y el borde.

Refinamiento Completo

La prueba de refinamiento completo consiste en refinar todos los cuadrantes de la malla 2D al mismo nivel. Debido al mayor uso de memoria para estas pruebas, se midió hasta $RL = 13$. En la Figura 5.3 se muestran unos ejemplos de las mallas resultante con estos parámetros.

(a) Nivel de Refinamiento 6 (b) Nivel de Refinamiento 13

Figura 5.3: Malla 2D resultante con refinamiento completo.

5.1.2. Resultados Obtenidos

Tiempo por Nivel de Refinamiento

Cuando se lleva a cabo el refinamiento, una operación clave es detectar si algún cuadrante vecino ya ha insertado algún nodo medio en alguno de los arcos en común. Para observar el impacto de esta propuesta respecto a esta tarea, se midió el tiempo de refinamiento en cada nivel de refinamiento hasta llegar al máximo parametrizado. En la Figura 5.4 se muestra la prueba de refinamiento de borde, tanto para la ejecución con la estructura de arcos como con la estructura de nodos en escala logarítmica. Este gráfico muestra que en cada RL, la estructura de datos del nodo mejoró el tiempo de cómputo, disminuyendo el tiempo que lleva ejecutar $RL = 20$ hasta en un 64 %.

La Figura 5.5 muestra el rendimiento de cada estructura de datos en cada nivel en el refinamiento para la región superior y el borde. Esto muestra que la estructura de datos del nodo mejoró el tiempo de cálculo en todas las pruebas. En la ejecución del nivel de refinamiento 14 la reducción del tiempo de ejecución alcanza el 68 %.

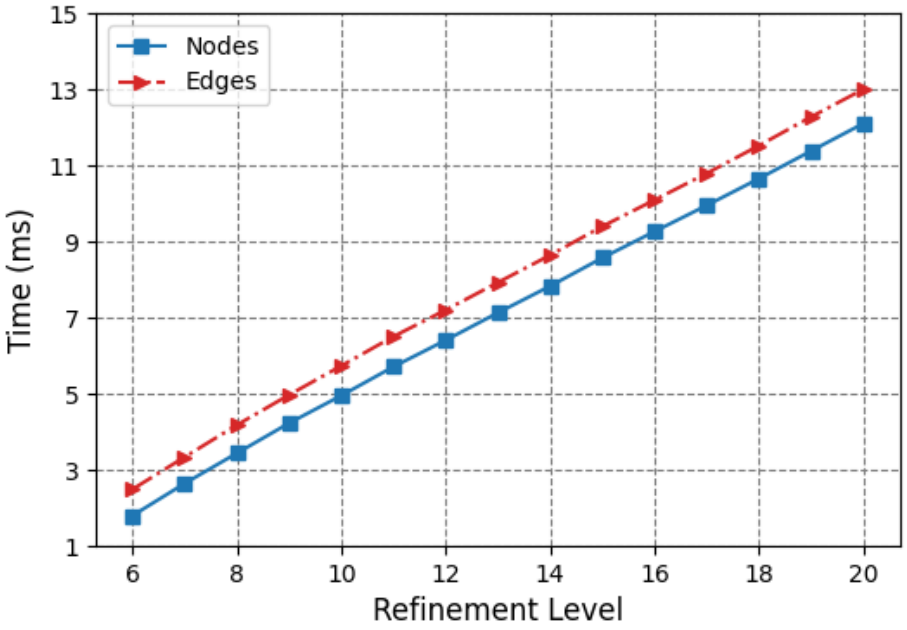


Figura 5.4: Tiempo por nivel en refinamiento de borde malla 2D.

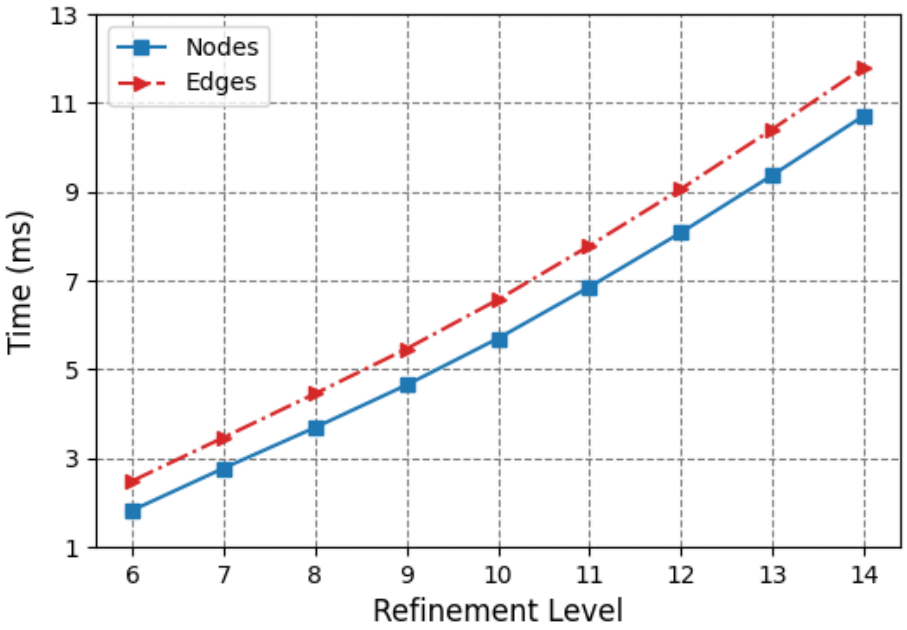


Figura 5.5: Tiempo por nivel en refinamiento de la región superior y el borde malla 2D.

Para la prueba de refinamiento completo de la polilínea, la Figura 5.6 muestra que, en cada *RL*, la estructura de datos del nodo mejoró el tiempo de cálculo, reduciéndolo hasta en un 70 % en la ejecución de $RL = 13$ en todos los cuadrantes.

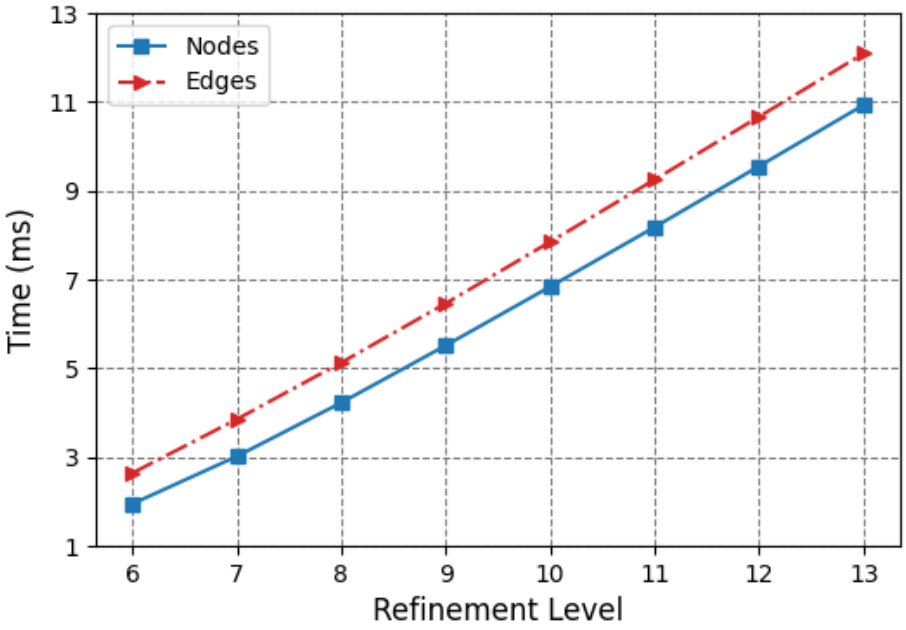


Figura 5.6: Tiempo por nivel en refinamiento de la región superior y el borde.

Tiempo de Refinamiento de Cuadrantes Desbalanceados

Para medir la capacidad de la estructura de nodos propuesta para detectar vecinos desbalanceados e identificar rápidamente el nodo medio de un arco, de forma análoga a como se mostró anteriormente para el proceso de refinamiento, medimos el tiempo que lleva refinar todos los cuadrantes que están desbalanceados en cada nivel. Esto se hizo para los tres tipos de refinamientos, hasta el nivel máximo que se podía alcanzar en cada tipo de prueba. Los resultados obtenidos se muestran en la Tabla 5.1. Tenga en cuenta que, como no se necesita balanceo para el caso de refinamiento completo, la diferencia entre las estructuras de nodo y de arcos es menos significativa.

Tabla 5.1: Tiempo de refinamiento cuadrantes desbalanceados (ms)

Nivel	Refinamiento de borde		Refinamiento de región superior y borde		Refinamiento Completo	
	Nodos	**Arcos**	**Nodos**	**Arcos**	**Nodos**	**Arcos**
6	0	0	0	0	0	0
7	1	2	0	1	0	0
8	3	6	2	6	0	0
9	7	15	4	12	0	0
10	15	31	9	24	0	0
11	41	98	43	98	0	0
12	59	119	70	137	0	0
13	143	305	200	367	0	0
14	263	532	496	899	-	-
15	748	1672	-	-	-	-
16	1206	2465	-	-	-	-
17	2295	4728	-	-	-	-
18	5048	10769	-	-	-	-
19	12607	28490	-	-	-	-
20	27195	61728	-	-	-	-

Tiempo Patrones de Transición

Además de calcular el tiempo de refinamiento para cuadrantes desbalanceados, otra forma de ver el impacto de la velocidad con la que se logra identificar los nodos intermedios de los arcos utilizando la estructura de datos del nodo, es medir el tiempo del proceso aplicación de los patrones de transición para las diferentes pruebas de refinamiento. Estos resultados se muestran en la Tabla 5.2.

Tabla 5.2: Tiempo patrones de transición malla 2D (ms)

Nivel	Refinamiento de borde		Refinamiento de región superior y borde		Refinamiento Completo	
	Nodos	**Arcos**	**Nodos**	**Arcos**	**Nodos**	**Arcos**
6	1	1	1	1	0	0
7	2	4	3	5	3	3
8	6	11	9	14	11	13
9	15	28	28	45	51	56
10	30	62	95	132	207	228
11	73	150	336	435	824	906
12	153	304	1247	1438	3285	3614
13	325	641	4656	5194	13131	14426
14	636	1284	17972	19509	-	-
15	1342	2742	-	-	-	-
16	2705	5497	-	-	-	-
17	5346	11154	-	-	-	-
18	10535	22255	-	-	-	-
19	20918	46397	-	-	-	-
20	44294	98290	-	-	-	-

Tiempo Total del Algoritmo

Junto con las métricas de tiempo anteriores, se midió el tiempo total de ejecución del proceso de generación de malla para diferentes configuraciones en las pruebas de refinamiento de borde, región superior y borde, y refinamiento completo para ver el impacto de esta propuesta en este algoritmo generador de mallas. Estos resultados se muestran en la Tabla 5.3.

Tabla 5.3: Tiempo total del algoritmo en mallas 2D (ms)

	Refinamiento de borde			**Refinamiento de región superior y borde**			**Refinamiento Completo**		
Nivel	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**
6	22	30	26.7 %	23	31	25.8 %	26	35	25.7 %
7	50	69	27.5 %	57	77	26.0 %	78	109	28.4 %
8	111	155	28.4 %	144	202	28.7 %	269	393	31.6 %
9	239	342	30.1 %	392	571	31.3 %	1042	1522	31.5 %
10	493	729	32.4 %	1170	1729	32.3 %	4104	6101	32.7 %
11	1093	1620	32.5 %	3917	5843	33.0 %	16135	24421	33.9 %
12	2223	3274	32.1 %	13745	20647	33.4 %	64981	99361	34.6 %
13	4585	6770	32.3 %	51232	78097	34.4 %	261740	405652	35.5 %
14	9032	13542	33.3 %	198661	306923	35.3 %	-	-	-
15	19136	28738	33.4 %	-	-	-	-	-	-
16	38419	57606	33.3 %	-	-	-	-	-	-
17	75510	114170	33.9 %	-	-	-	-	-	-
18	149222	229762	35.1 %	-	-	-	-	-	-
19	309977	484204	36.0 %	-	-	-	-	-	-
20	641616	996621	35.6 %	-	-	-	-	-	-

Uso de Memoria

Dado que esta propuesta se almacena en una *array* en lugar de un *Map*, como es el caso de la estructura de datos de arcos, el uso de la memoria es un aspecto interesante a considerar. Para medir el uso de memoria, se utilizó *valgrind*. En estas mediciones se consideró la memoria total utilizada por el algoritmo de mallado, utilizando cada una de las estructuras de datos que se comparan. Debido al mayor uso de memoria para la prueba de refinamiento completo de la polilínea, los datos se almacenaron hasta $RL = 12$. Los resultados obtenidos se muestran en la Tabla 5.4.

Tabla 5.4: Memoria Utilizada 2D (Gb)

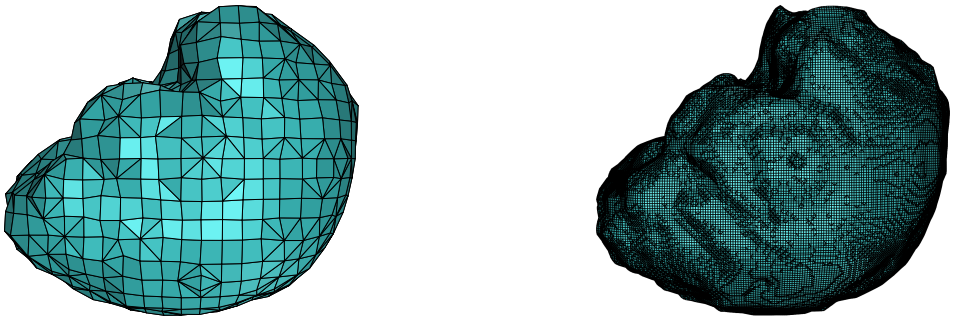
Nivel	Refinamiento de borde			Refinamiento de región superior y borde			Refinamiento Completo		
	Nodos	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**
6	0.001	0.001	0.0 %	0.001	0.001	0.0 %	0.001	0.001	0.0 %
7	0.002	0.002	0.0 %	0.002	0.003	33.3 %	0.003	0.004	25.0 %
8	0.004	0.005	20.0 %	0.006	0.007	14.3 %	0.012	0.016	25.0 %
9	0.008	0.010	20.0 %	0.016	0.021	23.8 %	0.048	0.058	17.2 %
10	0.015	0.021	28.6 %	0.053	0.063	15.9 %	0.191	0.228	16.2 %
11	0.033	0.045	26.7 %	0.175	0.219	20.1 %	0.698	0.897	22.2 %
12	0.065	0.087	25.3 %	0.642	0.789	18.6 %	2.792	3.543	21.2 %
13	0.130	0.175	25.7 %	2.469	3.006	17.9 %	11.060	14.173	22.0 %
14	0.249	0.338	26.3 %	9.771	11.704	16.5 %	-	-	-
15	0.524	0.707	25.9 %	-	-	-	-	-	-
16	1.049	1.396	24.9 %	-	-	-	-	-	-
17	2.040	2.791	26.9 %	-	-	-	-	-	-
18	3.973	5.476	27.4 %	-	-	-	-	-	-
19	8.053	10.845	25.7 %	-	-	-	-	-	-
20	17.073	23.408	27.1 %	-	-	-	-	-	-

5.2. Resultados Mallas 3D

En es esta sección se detallarán los tipos de refinamiento a los que se someterá la malla de superficie que representa la corteza cerebral humana, junto con los resultados obtenidos de las diferentes mediciones de los tiempos de ejecución y el uso de memoria.

5.2.1. Tipos de Refinamiento

Para el caso de las mallas 3D, los diferentes tipos de refinamiento que se les aplicarán generarán la misma superficie de la malla pero variará el interior de esta, dependiendo la configuración del refinamiento. En las Figuras 5.7a y 5.7b se muestran ejemplos de como sería la superficie de las mallas para los $RL = 4$ y $RL = 7$ respectivamente.



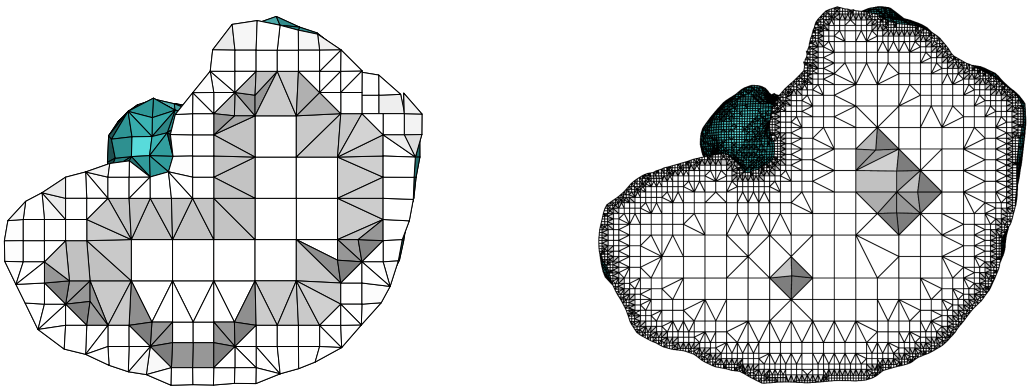
(a) Malla 3D resultante con $RL = 4$ (b) Malla 3D resultante $RL = 7$

Figura 5.7: Malla 3D resultante.

Dado lo anterior, los tipos de refinamiento que se realizarán para probar el desempeño de esta propuesta en mallas 3D son los siguientes:

Refinamiento de Borde

La prueba de refinamiento de borde consiste en refinar sólo los octantes que están en el borde de la malla de superficie de entrada, es decir, el contorno del dominio. Se contrastarán las estructuras de datos de nodos y arcos al ejecutar el algoritmo de mallado variando el parámetro RL del nivel 1 al 11. En la Figura 5.8 se muestran unos ejemplos de la mallas resultantes con estos parámetros.



(a) Malla 3D con corte en eje Z con $RL = 4$ (b) Malla 3D con corte en eje Z con $RL = 7$

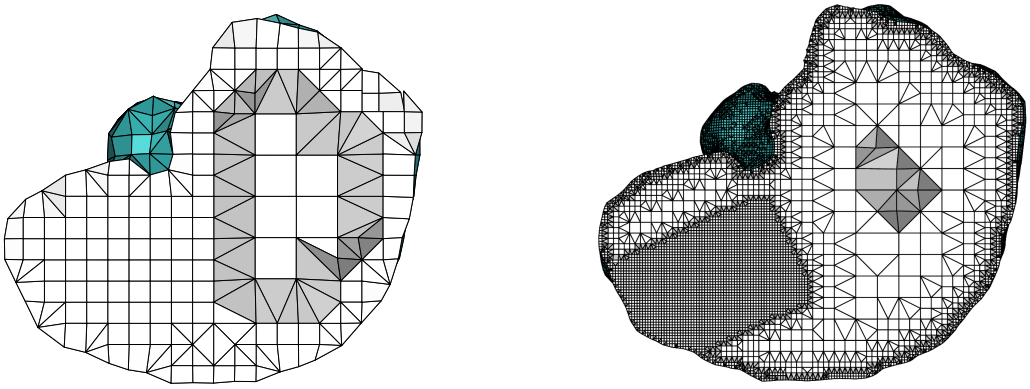
Figura 5.8: Malla 3D resultante con refinamiento de borde.

Refinamiento de Región 1 y de Borde

En el algoritmo de mallado definido en [10], es posible definir una región particular donde se puede especificar un RL diferente. Esta región está definida, para el caso 3D, por una malla de superficie que debe intersectar el dominio, entonces, en esta prueba, una región particular de la malla que se denominará "región 1", dicha región y el borde de la malla se refinarán al mismo nivel. Debido al mayor uso de memoria por parte de estas pruebas, se midió hasta $RL = 10$. En la Figura 5.2 se muestran unos ejemplos de las mallas resultantes con estos parámetros.

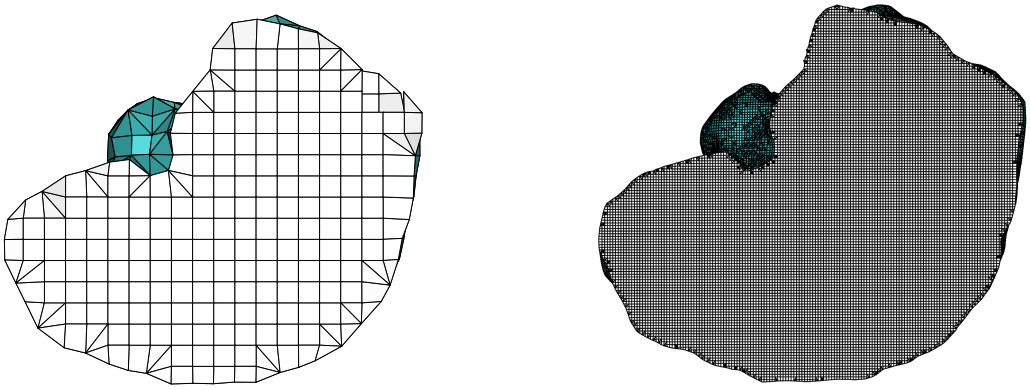
Refinamiento Completo

La prueba de refinamiento completo consiste en refinar todos los octantes de la malla 3D al mismo nivel. Debido al mayor uso de memoria para estas pruebas, se midió hasta $RL = 9$. En la Figura 5.3 se muestran unos ejemplos de las mallas resultante con estos parámetros.



(a) Malla 3D con corte en eje Z con $RL = 4$ (b) Malla 3D con corte en eje Z con $RL = 7$

Figura 5.9: Malla 3D resultante con refinamiento de región 1 y borde.



(a) Malla 3D con corte en eje Z con $RL = 4$ (b) Malla 3D con corte en eje Z con $RL = 7$

Figura 5.10: Malla 3D resultante con refinamiento completo.

5.2.2. Resultados Obtenidos

Tiempo por Nivel de Refinamiento

Para observar el impacto de esta propuesta respecto a su capacidad de revisar los arcos al momento de la refinación, se midió el tiempo de refinamiento para cada *RL* hasta llegar al máximo parametrizado. En la Figura 5.11 se muestra la prueba de refinamiento de borde, tanto para la ejecución con la estructura de arcos como con la estructura de nodos en escala logarítmica. Este gráfico muestra que en cada *RL*, la estructura de datos del nodo mejoró el tiempo de cómputo, disminuyendo el tiempo que lleva ejecutar $RL = 11$ hasta en un 15.8 %.

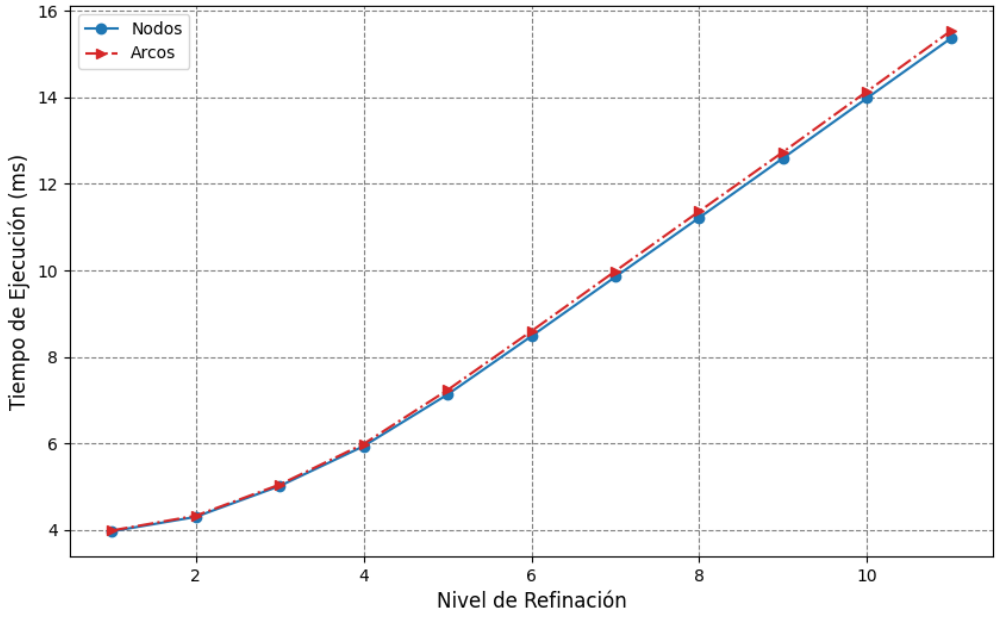


Figura 5.11: Tiempo por nivel en malla 3D con refinamiento de borde.

La Figura 5.12 muestra el rendimiento de cada estructura de datos en cada nivel en el refinamiento para la región 1 y el borde. Esto muestra que la estructura de datos del nodo mejoró el tiempo de cálculo en todas las pruebas. En la ejecución del nivel de refinamiento 10 la reducción del tiempo de ejecución alcanza el 27.5 %.

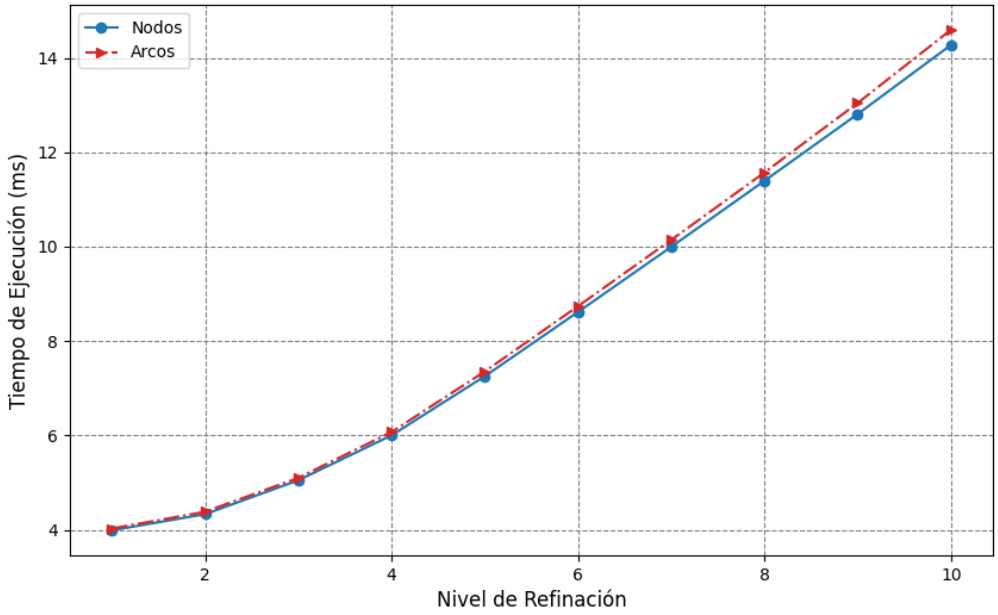


Figura 5.12: Tiempo por nivel en malla 3D con refinamiento de región 1 y borde.

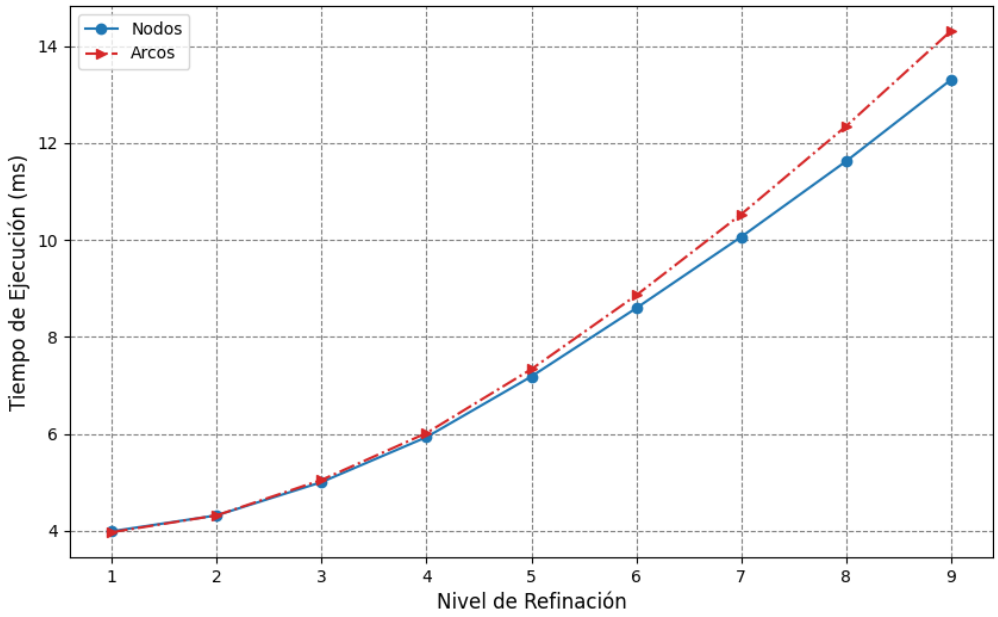


Figura 5.13: Tiempo por nivel en malla 3D con refinamiento completo.

Para la prueba de refinamiento completo de la malla de superficie, la Figura 5.13 muestra que, en cada *RL*, la estructura de datos del nodo mejoró el tiempo de cálculo, reduciéndolo hasta en un 63.5 % en la ejecución de $RL = 9$.

Tiempo de Refinamiento de Octantes Desbalanceados

Para medir la capacidad de nuestra estructura de nodos para detectar vecinos desbalanceados e identificar rápidamente el nodo medio de un arco, de forma análoga a como se mostró anteriormente para el proceso de refinamiento, medimos el tiempo que lleva refinar todos los cuadrantes que están desbalanceados en cada nivel. Esto se hizo para los tres tipos de refinamientos, hasta el nivel máximo que se podía alcanzar en cada tipo de prueba. Los resultados obtenidos se muestran en la Tabla 5.5. Tenga en cuenta que, como no se necesita balanceo para el caso de refinamiento completo, la diferencia entre las estructuras de nodo y de arcos es menos significativa.

Tabla 5.5: Tiempo de refinamiento octantes desbalanceados (ms)

	Refinamiento de borde		**Refinamiento de región 1 y borde**		**Refinamiento Completo**	
Nivel	**Nodos**	**Arcos**	**Nodos**	**Arcos**	**Nodos**	**Arcos**
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	4	10	3	9	0	0
6	34	77	34	83	0	0
7	170	399	196	490	0	0
8	786	1879	863	2129	0	0
9	3373	8308	3923	9677	0	0
10	14483	36264	17005	42638	-	-
11	62544	153501	-	-	-	-

Tiempo Patrones de Transición

Junto con calcular el tiempo de refinamiento para los octantes desbalanceados, otra forma de ver el impacto de la velocidad con la que se logra identificar los nodos intermedios de los arcos utilizando la estructura de datos del nodo, es medir el tiempo del proceso aplicación de los patrones de transición para las diferentes pruebas de refinamiento. Estos resultados se muestran en la Tabla 5.6.

Tabla 5.6: Tiempo patrones de transición malla 3D (ms)

	Refinamiento de borde		**Refinamiento de región 1 y borde**		**Refinamiento Completo**	
Nivel	**Nodos**	**Arcos**	**Nodos**	**Arcos**	**Nodos**	**Arcos**
3	4	5	4	4	1	1
4	23	25	20	22	4	5
5	119	139	123	144	24	29
6	596	731	654	829	193	243
7	2738	3359	3172	3993	1471	1889
8	11675	14505	14175	17984	11507	14098
9	47797	61207	66017	85500	90934	114534
10	193835	250867	322767	416202	-	-
11	785481	1022739	-	-	-	-

Tiempo Total del Algoritmo

Junto con las métricas de tiempo anteriores, se midió el tiempo total de ejecución del proceso de generación de malla para diferentes configuraciones en las pruebas de refinamiento de borde, región superior y borde, y refinamiento completo para ver el impacto de esta propuesta en este algoritmo generador de mallas 3D. Estos resultados se muestran en la Tabla 5.7.

Tabla 5.7: Tiempo total del algoritmo en mallas 3D (ms)

	Refinamiento de borde			Refinamiento de región 1 y borde			Refinamiento Completo		
Nivel	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**
3	488	499	2.2 %	504	515	2.1 %	485	488	0.6 %
4	1006	1040	3.3 %	1060	1107	5.9 %	995	1038	4.1 %
5	2787	2968	6.1 %	3084	3299	6.5 %	3006	3220	6.6 %
6	9655	10657	9.4 %	11102	12474	11.0 %	12726	14675	13.3 %
7	37644	41913	10.2 %	44878	51301	12.5 %	66407	83227	20.2 %
8	147219	166265	11.5 %	187392	218819	14.4 %	406917	546412	25.5 %
9	584197	664672	12.1 %	829896	1012896	18.1 %	2767026	4411712	37.3 %
10	2385763	2730240	12.6 %	4000014	5204379	23.1 %	-	-	-
11	9652587	11263654	14.3 %	-	-	-	-	-	-

Uso de Memoria

Junto con los resultados anteriores, otro punto que permite observar el impacto de esta propuesta en las mallas volumétricas es el uso de la memoria. Para medir el uso de memoria, se utilizó nuevamente *valgrind*. En estas mediciones se consideró la memoria total utilizada por el algoritmo de mallado, obteniendo los valores en cada una de las estructuras de datos que se comparan. Los resultados obtenidos se muestran en la Tabla 5.8.

Tabla 5.8: Memoria Utilizada 3D (Gb)

	Refinamiento de borde			**Refinamiento de región 1 y borde**			**Refinamiento Completo**		
Nivel	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**	**Nodos**	**Arcos**	**Mejora**
3	0.004	0.004	0 %	0.004	0.004	0 %	0.004	0.004	0 %
4	0.006	0.007	14.3 %	0.006	0.007	14.3 %	0.006	0.007	14.3 %
5	0.020	0.025	20.0 %	0.023	0.028	17.9 %	0.027	0.038	28.9 %
6	0.087	0.103	15.5 %	0.103	0.125	17.6 %	0.199	0.251	20.7 %
7	0.359	0.429	16.3 %	0.479	0.596	19.6 %	1.503	1.825	17.6 %
8	1.503	1.717	12.5 %	2.576	3.006	14.3 %	11.811	14.281	17.3 %
9	5.905	7.087	16.7 %	13.744	16.428	16.3 %	-	-	-
10	23.837	28.562	16.5 %	-	-	-	-	-	-

Capítulo 6

Conclusiones

En el presente trabajo se realizó un estudio y análisis de un algoritmo de generación de mallas de superficie y de volumen basadas en *quadtree* y *octree* respectivamente. En este algoritmo se necesita almacenar dinámicamente los vecinos de cada celda para mantener el balance en la malla. A partir de la problemática en el almacenamiento de la información, se han presentado diferentes alternativas para esta tarea, utilizando estructuras basadas en árboles o en los arcos de la malla, donde siempre se busca disminuir el tiempo que toma el actualizar y obtener información por medio de estas estructuras de datos. Dado lo anterior, se planteó la hipótesis de que era posible crear una estructura de datos basada en los nodos de la malla que permitiera obtener información respecto a sus celdas vecinas e identificar el nodo medio de un arco en tiempo constante, de forma que disminuya el tiempo total de generación de mallas 2D y 3D basadas en *quadtree/octree*. Para lograr comprobar la hipótesis, se propuso una modificación del algoritmo analizado, utilizando la estructura de datos propuesta.

La primera etapa desarrollada consistió en estudiar y considerar las características de las propuestas actuales en el estado del arte para determinar sus desventajas. En el caso de las basadas en árboles, la complejidad de la navegación para la búsqueda de datos y la acumulación de celdas no útiles como nodos intermedios, son las principales desventajas de este enfoque. Pese a que se han propuesto alternativas que prometen buenos resultados, su posible adaptación para mallas 3D es bastante cuestionable. Para las alternativas enfocadas en

almacenar la información en los arcos, en particular, en la propuesta utilizada por el algoritmo estudiado, el uso de un *Map* para el almacenamiento de los datos de los arcos provoca que su desempeño esté limitado al tiempo que toma operar sobre él. Sumado a esto, el uso de punteros por parte del *Map* lo hace una alternativa costosa ante el creciente número de celdas al aumentar el parámetro de refinación. En último lugar, no se encontraron propuestas actuales que utilicen en particular los nodos para almacenar la información en una malla tipo *quadtree/octree*, solo alternativas que podían representar elementos de la malla desde algún nodo en particular.

A partir de lo anterior, se estudió lo expresado en la conjetura, donde se percató que el número de nodos es siempre menor o igual al de arcos en la generación de cuadrantes y octantes. En base a esto, la siguiente etapa fue buscar una forma de obtener tiempo constante al acceso a los datos que almacenarían los nodos. En esa línea y aprovechando el hecho que cada nodo tiene un identificador único en la malla, se decidió utilizar un *Array* o vector que permitiera, a partir de un índice, obtener los datos de la posición solicitada en tiempo $O(1)$. Una vez determinada la estructura que almacenaría a los datos de los nodos, se implementó para el caso 2D, donde se definió la información que debía almacenar cada nodo junto con la forma en la que se puede detectar la presencia de un nodo intermedio de un arco y distinguirlo en los casos que existan más de un nodo en dicho arco. Una vez lograda la implementación y obtención de resultados para la versión 2D, estos fueron presentado en la conferencia *SCCC 2023*, en el marco del proceso requerido para la presentación de este trabajo, lo que se puede evidenciar en [19]. Posterior a esto, se implementó la versión 3D de la propuesta, con la ligera diferencia en la forma en que se identifican los nodos medios de un arco, debido a la complejidad algorítmica que se genera intentar identificarlos con los niveles de refinación de las celdas vecinas en 3D.

6.1. Sobre los Resultados Obtenidos

El objetivo principal de este trabajo fue acelerar el proceso de balanceo. Con este propósito, se diseñó una estructura de datos basada en nodos bajo el proceso anteriormente mencionado, lo que permitió mantener la información de los vecinos de las celdas. Esto último es crucial

en el proceso de refinamiento de vecinos desbalanceados.

Se llevaron a cabo tres tipos de pruebas diferentes para determinar la eficiencia en términos de tiempo computacional y uso de memoria, tanto para las mallas 2D como para las 3D. Para las mallas de dos dimensiones se realizaron las pruebas de refinamiento de borde, refinamiento de la región superior y el borde, y refinamiento completo. En el primero, solo se refinaron los cuadrantes que intersectan el límite del dominio, lo que requirió más operaciones de balanceo. En el segundo caso, el límite del dominio y toda la región en la parte superior del dominio tenían el mismo nivel de refinamiento, lo que implicó menos operaciones de balance debido a que toda la región superior ya estaba balanceada. Y finalmente, el refinamiento completo, lo que significa que no se necesitaron operaciones de balance en absoluto. Para los tres casos, se contrastaron los resultados con la estructura de datos basada en arcos propuesta en [10].

Con respecto al proceso de balanceo mostrado en la Tabla 5.1, como era de esperar, la reducción es significativa y puede llegar a más del 50 % en el caso del refinamiento en el borde; sin embargo, este impacto es menos significativo si se necesita menos balanceo de cuadrantes, como en el caso del refinamiento completo. Nuevamente, al igual que durante el proceso de refinamiento, al aplicar los patrones de transición también necesitamos verificar la inserción de nodos intermedios; por lo tanto, la Tabla 5.2 muestra resultados equivalentes a las Figuras 5.4, 5.5 y 5.6.

Como último punto de los resultados de tiempo computacional obtenidos en las mallas 2D, en la Tabla 5.3 se muestra que, en todos los casos, la estructura propuesta ayuda a reducir el tiempo total. Por supuesto, esta reducción será más importante para niveles de refinamiento más altos y cuando el proceso de balanceo es significativo, es decir, cuando hay más diferencia en el *RL*. Al final, para la malla más refinada en los tres casos de prueba, esta propuesta utiliza, aproximadamente, solo $\frac{2}{3}$ del tiempo original. En cuanto a la memoria, la Tabla 5.4 muestra que al aumentar el *RL*, esta propuesta lograr reducir entre un 15 % a un 25 % el uso de memoria en los niveles más altos de *RL*. Además, para los niveles más bajos de *RL*, es importante destacar que esta propuesta nunca utiliza más memoria que la estructura de datos basada en arcos.

Respecto a los experimentos ejecutados sobre las mallas 3D, estos fueron el refinamiento de borde, refinamiento de una región de interés de la malla y el borde, y el refinamiento completo. Para estos tres tipos de refinamiento se contrastó los resultados con la estructura de datos basada en arcos propuesta en [10].

Con respecto al proceso de balanceo mostrado en la Tabla 5.5, al igual que en el caso en dos dimensiones, la reducción es significativa y puede llegar a más del 50 %, como sucede en el refinamiento en el borde. Al igual que el caso 2D, la propuesta no afecta los tiempos en el refinamiento completo dado que en este caso no hay octantes por balancear.

Respecto al impacto de la propuesta en la generación de los patrones de transición, donde también se necesita verificar la existencia de nodos intermedios, la propuesta redujo el tiempo para todos los casos, llegando hasta más de un 23 % como es en el caso del refinamiento de borde, donde se podría inferir que hay una mayor cantidad de patrones por general, debido a que los octantes que están en el centro están a un nivel de refinamiento menor que le borde.

En relación con el tiempo que toma refinar cada nivel, se puede evidenciar un comportamiento diferente al caso 2D. En las mallas de superficie, era predecible que en refinamiento de borde en 2D, la propuesta mejoraba los tiempos de ejecución por niveles debido a que hay más que balancear y menos cuadrantes que refinar. Sin embargo, como se puede ver comparando los Gráficos 5.11, 5.12 y 5.13, el caso donde se nota una mayor reducción del tiempo computacional, es en el refinamiento completo de la malla volumétrica. Esto se puede producir debido a que para estos experimentos, a diferencia de lo ocurrido en 2D, no se pudo llegar hasta un nivel mucho más alto en el refinamiento de borde, debido a la gran cantidad de recursos que consumen estas pruebas. Por otro lado, una posible hipótesis para este comportamiento, es que en el refinamiento de borde, como solamente se refinan los octantes que intersectan con el contorno del dominio, la cantidad de octantes a refinar no es tan grande como el caso del refinamiento completo. Esto explicaría que la estructura de datos basada en nodos reduce el tiempo que toma el refinamiento de cada nivel en el refinamiento de borde, pero esta reducción crece a un ritmo más lento que en los otros tipos de refinamiento, llegando a un 15 % para el nivel 11 mientras que el refinamiento completo llega a reducir en más de un 60 % el tiempo computacional.

En los tiempos totales de ejecución del algoritmo generador de mallas 3D, la propuesta logra reducirlo y obtiene resultados similares respecto a los obtenidos para mallas 2D. Sin embargo, debido a lo mencionado anteriormente, en este caso la mayor reducción se da en el caso de las mallas de refinamiento completo, llegando hasta un 37 %, superando lo obtenido para las mallas en dos dimensiones. Por otro lado, para el refinamiento de borde, por lo que se comentaba anteriormente, esta reducción llegó hasta un 15 % del tiempo total, lo que discrepa con el más de 25 % que se obtuvo para el mismo tipo de refinamiento en 2D.

En último lugar, respecto a la memoria utilizada para el caso 3D, como era de esperarse el algoritmo utiliza muchos más recursos en comparación al caso 2D para los mismos niveles de refinamiento. En este caso, la propuesta de este trabajo logra en todas las pruebas ocupar a lo más los mismos recursos que la estructura de arcos y logra una reducción de hasta un 28 %. En promedio, la reducción de memoria provocada por esta propuesta es de un 15 %. Cabe mencionar, que para las pruebas más altas no se pudo medir la memoria utilizada por la estructura de arcos ya que las características del *hardware* no lo permitieron. Sin embargo, la estructura de nodos pudo rescatar un uso de más de 73 GB para máximos niveles de refinamiento de cada prueba.

En resumen, los resultados de este trabajo logran evidenciar que la nueva estructura de datos propuesta ayuda, en varios pasos del algoritmo, reducir significativamente el tiempo de generación de una malla de *quadtree/octree* balanceada, cumpliendo con la hipótesis y los objetivos fijados para este trabajo. Además, al hacerlo, logra disminuir el uso de memoria en todas las pruebas, tanto en mallas 2D y 3D, en comparación a la anterior estructura de datos basada en arcos que se empleaba con este propósito.

6.2. Cumplimiento de los Objetivos y Trabajo Futuro

Respecto a los objetivos propuestos en este trabajo, se logró definir e implementar una estructura de datos basada en los nodos de la malla. Además, esta estructura almacena la información de sus vecinos y permite verificar la presencia de un nodo intermedio de un arco en tiempo $O(1)$. De esta forma, a partir de todos los resultados expuestos en este trabajo, se

logró disminuir el tiempo global de la creación de mallas geométricas, cumpliendo de esta manera los dos objetivos generales de este trabajo.

Junto con lo anterior, esta propuesta logra, para todos los casos, utilizar a lo más la misma memoria que la estructura de arcos y en la mayoría de los casos logra una reducción de recursos a utilizar de un 15 %, lo que permite concluir que se cumplieron todos objetivos específicos definidos. En base a esto y recordando que se consiguió almacenar esta estructura en un *Array* que aseguraba tiempo constante al acceso de la información, se pudo probar la hipótesis que mediante una estructura de datos asociada a los nodos de una malla *quadtree/octree*, es posible reducir el tiempo de búsqueda de las celdas vecinas y la obtención de nodos intermedios de un arco en una celda de $O(Log(n))$ a $O(1)$.

Dado que en este trabajo se probó que esta alternativa logra optimizar el proceso de generación de mallas balanceadas, esto abre varias posibles vías de investigación sobre esta propuesta. En primer lugar, se podría diseñar una primera versión paralela del algoritmo generador de mallas presentado en [10], utilizando la estructura de datos nodos propuesta.

Junto con lo anterior, se podría estudiar si esta estructura de datos permite reabrir el debate respecto a si es mejor balancear en cada nivel de refinamiento o al final de la generación de la malla. En caso que esto último obtenga mejores resultados para el caso desbalanceado, también se podrían estudiar respecto a algoritmos paralelos de mallado por regiones de interés que utilicen la estructura de datos de nodos.

Otra posible investigación, es la utilidad de esta propuesta en la construcción de QUADTREE desbalanceados, que son ocupados en el área de visión para la generación de rutas con drones o vehículos inteligentes. Hoy en día, estos utilizan mayoritariamente la estructuras basadas en árboles, ya que no importa mantener el balance entre las celdas y solamente se refina cuando hay una región de interés. Se podría estudiar si esta propuesta optimiza los tiempos de generación del *quadtree* bajo las condiciones anteriormente mencionadas.

Como última posible vía de investigación futura, quedaría la opción de estudiar como adaptar está estructura de nodos a mallas que no se basen en el *quadtree/octree*, como lo son las mallas por características, mallas basadas en la triangulación de *delaunay*, entre otras.

Bibliografía

[1] P. L. George and P. Frey, *Mesh generation: application to finite elements*. John Wiley & Sons, 2013.

[2] M. Deering, "Geometry compression," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 13–20, 1995.

[3] W. Lijuan, "An integrative optimization algorithm of quadrilateral meshes based on cloud data," in *2009 Chinese Control and Decision Conference*, pp. 1932–1936, 2009.

[4] S. Hang, "Tetgen, a delaunay-based quality tetrahedral mesh generator," *ACM Trans. Math. Softw*, vol. 41, no. 2, p. 11, 2015.

[5] Y. Ito, A. M. Shih, and B. K. Soni, "Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates," *International Journal for Numerical Methods in Engineering*, vol. 77, no. 13, pp. 1809–1833, 2009.

[6] C. Lobos and E. González, "Mixed-element octree: a meshing technique toward fast and real-time simulations in biomedical applications," *International journal for numerical methods in biomedical engineering*, vol. 31, no. 12, 2015.

[7] A. Botella, B. Lévy, and G. Caumon, "Indirect unstructured hex-dominant mesh generation using tetrahedra recombination," *Computational Geosciences*, vol. 20, no. 3, pp. 437–451, 2016.

[8] J. Elmesbahi, O. Bouattane, and Z. Benabbou, "O(1) time quadtree algorithm and its application for image geometric properties on a mesh connected computer (mcc)," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 12, pp. 1640–1648, 1995.

[9] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[10] F. Jaillet and C. Lobos, "Fast quadtree/octree adaptive meshing and re-meshing with linear mixed elements," *Engineering with Computers*, vol. 38, no. 4, pp. 3399–3416, 2022.

[11] M. Yerry and M. Shephard, "A modified quadtree approach to finite element mesh generation," *IEEE Computer Graphics and Applications*, vol. 3, no. 01, pp. 39–46, 1983.

[12] H. Samet and C. A. Shaffer, "A model for the analysis of neighbor finding in pointer-based quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 717–720, 1985.

[13] D. Eppstein, M. T. Goodrich, and J. Z. Sun, "The skip quadtree: a simple dynamic data structure for multidimensional data," in *Proceedings of the twenty-first annual symposium on Computational geometry*, pp. 296–305, 2005.

[14] H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer graphics and image processing*, vol. 18, no. 1, pp. 37–57, 1982.

[15] K. Aizawa and S. Tanaka, "A constant-time algorithm for finding neighbors in quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 7, pp. 1178–1183, 2008.

[16] K. Aizawa, K. Motomura, S. Kimura, R. Kadowaki, and J. Fan, "Constant time neighbor finding in quadtrees: An experimental result," in *2008 3rd International Symposium on Communications, Control and Signal Processing*, pp. 505–510, IEEE, 2008.

[17] S. Salinas-Fernández, N. Hitschfeld-Kahler, A. Ortiz-Bernardin, and H. Si, "Polylla: polygonal meshing algorithm based on terminal-edge regions," *Engineering with Computers*, vol. 38, no. 5, pp. 4545–4567, 2022.

[18] M. Kallmann and D. Thalmann, "Star-vertices: a compact representation for planar meshes with adjacency information," *Journal of Graphics Tools*, vol. 6, no. 1, pp. 7–18, 2001.

[19] J. Díaz, C. Lobos, and N. Hitschfeld-Kahler, "Node-based data structure for balancing process optimization of quadtree meshes," in *2023 42nd IEEE International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–7, 2023.