Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

SIMULACIÓN LIBRE DE MALLAS POR PARTÍCULAS SUAVIZADAS OPTIMIZADO MEDIANTE OCTREE DINÁMICO EN CUDA

T E S I S

QUE PARA OPTAR POR EL GRADO DE

Doctor en Ciencia e Ingeniería de la Computación

PRESENTA:

David Arturo Soriano Valdez

Tutores Principales:

Dr. Fernando Arámbula Cosío, IIMAS

Dr. Alfonso Gastélum Strozzi, ICAT

México, Cd. Mx., Agosto 2021

Agradecimientos

Agradezco a mis familiares y amigos por su amor, amistad y apoyo.

Agradezco al posgrado en ciencias e ingeniería de la computación, así como a mis tutores: Dr. Alfonso Gastelum Strozzi, Dr. Fernando Arámbula Cosío y el Dr. Miguel Ángel Padilla Castañeda, por todo el apoyo en mi trabajo de investigación.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico mediante la beca otorgada con número (CVU/Becario): 559010/296666, para realizar los estudios de Doctorado en Ciencia e Ingeniería de la Computación.

Agradezco al Dr. Patrice Delmas y el equipo del IVSLab de la Universidad de Auckland por haberme permitido trabajar con ellos durante la estancia de investigación que se realizó en el periodo de Mayo a Agosto de 2019.

Resumen

En este trabajo se aborda el problema específico de la simulación libre de mallas con métodos de partículas suavizadas. En específico se hace énfasis en el problema del cálculo de vecindades y se presenta un método novedoso para mejorar el desempeño de dicho problema. En vista de que el problema puede ser resuelto con diferentes enfoques, se delimita el problema para el caso específico en el que se desea aprovechar el poder de cómputo que proporcionan las tarjetas gráficas actuales.

El método propuesto emplea un Octree indexado y ordenado mediante claves Morton, este método es ideal para aplicarlo a problemas que pueden ser representados con una serie de puntos en el espacio en tres dimensiones, por ejemplo, una nube de puntos.

El método propuesto en este trabajo tiene la ventaja que mediante operaciones binarias es capaz de determinar todos los elementos que se encuentran en su vecindad, esto lo hace sumamente eficiente al ser implementado en arquitecturas de cómputo en paralelo como lo es la arquitectura de las tarjetas gráficas de Nvidia, CUDA.

Es debido a estas características que estos métodos tienen aplicación en varias ramas del modelado de datos y simulación de fenómenos, entre estos uno popular es la simulación mediante el método de Hidrodinámica de partículas suavizadas (SPH).

En este trabajo se presenta además de la implementación del método y las pruebas de su eficiencia y tiempos de ejecución en comparación con otros, una implementación del método de SPH empleado para simular un fenómeno de flujo y cuerpos visco elásticos, un método nuevo para resolver colisiones entre fluidos y solidos utilizando un campo de normales y una implementación para cálculo de características en superficies de objetos representados por nubes de puntos.

El uso de métodos en paralelo hace posible que el método presentado se aplique en condiciones donde lo que se busca no es realismo si no una solución que pueda correr a 60 cuadros por segundo para su uso en medios interactivos como videojuegos o simuladores serios.

La alta cantidad de partículas usadas en las diferentes simulaciones y análisis de superficie representa un reto para la presentación de resultados claros, visuales y estadísticos, esto hace necesario implementar técnicas de *Big Data* que permitan dar al usuario un resultado claro de interpretar.

Índice de figuras

1.1. Imagen tomada de Dal Ferro et al. [41]. Mapa de color de estructura conductora de agua en subsuelo. 9

2.1. División espacial mediante el uso de nodos contenidos en un Octree a diferentes niveles de profundidad. 15

2.2. Relación entre nodos padre y nodos hijo en el Octree. 15

2.3. Diagrama de trama binaria que compone la clave Morton (MK). 16

2.4. Construcción de clave Morton (MK) usando entrelazado binario a partir de posición. 17

2.5. A) Clave Morton parcial (subMK) para nodos hijo dentro de un nodo padre. B) Secuencia de indexado empleando la curva z-order en 3 dimensiones. 19

2.6. A) Etapas de cálculo de vecindades sin ordenamiento. B) Etapas de cálculo de vecindades empleando ordenamiento por MK. 20

2.7. Diagrama de apuntadores necesarios en nodos para enlazar con el sistema de partículas sin un ordenamiento especifico. 21

2.8. Diagrama de apuntadores necesarios en nodos para enlazar con el sistema de partículas ordenando las partículas mediante su clave Morton (MK). 22

2.9. Vecindad radial de un nodo (naranja). Los nodos vecinos (verde) son todos los nodos adyacentes por cara, arista o vértice. 23

2.10. Conjuntos de nodos que conforman la vecindad empleada por el método. A) Vecinos por cara. B) Vecinos por arista. C)Vecinos por vértice. D) Vecindad radial empleada por el método. 24

3.1. Aproximación mediante suma de partículas. El kernel $W(r - r', h)$ es usado para calcular el valor de la propiedad de campo para la partícula i mediante la suma ponderada de cada partícula j que está contenida en la región S. 31

3.2. Gráficas de kernels W, en rojo W, en azul ∇W, en rojo $\nabla^2 W$. A) Kernel poly6. B) Kernel spiky. C) Kernel visc. 37

3.3. Volumen de un nodo siendo ocupado en su totalidad por 8 partículas. . 38

4.1. Modelos de deformación viscoelásticos. A) Maxwell. B) Kelvin-Voigt. C) SLS. 42

4.2. Diagrama de resultado de colisión sobre una superficie donde se considera la normal a la superficie $\hat{n}$, dirección y ángulo de incidencia, $\vec{v_1}$ y Θ_1; esto da como resultado una dirección y ángulo de reflejo, $\vec{v_2}$ y Θ_2. . 45

4.3. Escenario de detección y no detección de colisiones entre partículas. Hay colisión cuando el producto escalar es menor a 0, mientras que con un valor igual o mayor que 0 no se considera la existencia de colisión. . . . 47

5.1. Ejemplo de simulación de flujo en tubo, presentada con Octree a diferente nivel de profundidad. A) Nivel 9 con despliegue de 141,114 elementos. B) Nivel 8 con despliegue de 117,209 elementos. C) Nivel 7 con despliegue de 11,503 elementos. 51

5.2. Ejemplo de simulación de flujo en tubo empleando visualización de datos de presión en fluido con metodología lagrangiana. 51

5.3. Aplicación de medida de forma, empleando el valor de cambio de normal asociada a la superficie, usando kernels de SPH con diferentes niveles de resolución de Octree. 52

6.1. Diagramas UML de estructuras de datos implementadas para las pruebas de cálculo de vecindad. A) LUT. B) Alternative. C) Opt. D) Recursive. 54

6.2. Diagrama de flujo del proceso usado para el calculo de vecindad. 56

6.3. Escenarios empleados para realización de experimentos. A) Nube de puntos aleatoria (500,000 partículas). B) Nube de puntos aleatoria (1,000,000 partículas). C) Modelo de ramas y bifurcaciones (495,839 partículas). . 57

6.4. Gráfica de rendimiento de implementaciones LUT. Alt, Rec y Opt para escenario (A) empleando Octree con nivel de profundidad 7,8 y 9. . . . 59

6.5. Gráfica de rendimiento de implementaciones LUT. Alt, Rec y Opt para escenario (B) empleando Octree con nivel de profundidad 7,8 y 9. . . . 59

6.6. Gráfica de rendimiento de implementaciones LUT. Alt, Rec y Opt para escenario (C) empleando Octree con nivel de profundidad 7,8 y 9. . . . 60

6.7. Diagrama de memoria usada por implementaciones LUT, Alternative, Recursive y Opt. Diagramas generados mediante Nvidia Nsight para Microsoft Visual Studio. 61

6.8. Diagrama de memoria usada por implementaciones LUT, Alternative, Recursive y Opt. Diagramas generados mediante Nvidia Nsight para Microsoft Visual Studio. 63

6.9. Gráfica de comparativa de desempeño entre implementaciones OPT y SORTED con un Octree con nivel máximo de profundidad 7. Se usaron 3 escenarios y 3 GPUs diferentes. 65

6.10. Gráfica de comparativa de desempeño entre implementaciones OPT y SORTED con un Octree con nivel máximo de profundidad 8. Se usaron 3 escenarios y 3 GPUs diferentes. 66

6.11. Gráfica de comparativa de desempeño entre implementaciones OPT y SORTED con un Octree con nivel máximo de profundidad 9. Se usaron 3 escenarios y 3 GPUs diferentes. 67

6.12. Diagrama con perfil de velocidades parabólico de fluido laminar dentro de un tubo. 69

6.13. Prueba 1. Resultado de simulación (puntos) y analítico (línea) de fluido laminar. 69

6.14. Prueba 2. Resultado de simulación (puntos) y analítico (línea) de fluido laminar. 70

6.15. Prueba 3. Resultado de simulación (puntos) y analítico (línea) del flujo laminar. 70

6.16. Prueba 3. Resultado de simulación del flujo laminar. 71

6.17. Corte transversal de cilindro, se observan errores en la aproximación de la forma debido a la discretización en voxeles (píxeles). 72

6.18. Resultado de simulación de flujo donde se gráfica la velocidad promedio de las partículas dentro de un cilindro, se toma como referencia los planos centrales del cilindro. 73

6.19. Velocidad de partículas en cilindro, los resultados muestran concordancia con la solución analítica. 74

6.20. Gráfica de desempeño obtenido para los procesos UND y UNF, en escenarios de nubes de puntos aleatorias. Se utilizó la implementación Opt y un Octree con nivel de profundidad máxima 8. 77

6.21. Secuencia de eventos de simulación de flujo con objeto estático. A) Escenario con 21,288 partículas. B) Escenario con 145,777 partículas. . . . 79

6.22. Secuencia de eventos de simulación de flujo con objeto deformable. Escenario con 21,288 partículas. 81

6.23. Desempeño por cuadro para cada uno de los escenarios usando un Octree con nivel de profundidad máxima 7. 83

6.24. Desempeño por cuadro para cada uno de los escenarios usando un Octree con nivel de profundidad máxima 8. 84

6.25. Desempeño por cuadro para cada uno de los escenarios usando un Octree con nivel de profundidad máxima 9. 84

6.26. Secuencia de eventos de simulación con implementación de cálculo de vecindad SORTED. Escenarios A, B Y C. 85

6.27. Diagrama de flujo para creación de modelos de partículas a partir de modelos de mallas triangulares. 85

6.28. Ejemplos de modelos de mallas triangulares a modelos de partículas. . . 86

6.29. Resultado gráfico de experimento de estudio de forma con SPH. Modelo con 1,604,254 partículas. A) Nivel 6. B) Nivel 7 88

6.30. Estructuras de poros que forman el volumen. Arriba a la izquierda: Todos los poros 3D dentro del volumen. Arriba a la derecha: dimensión de los poros del tamaño 0 a 100 voxeles. Abajo a la izquierda: Tamaño de los poros de 100 a 1000 voxeles. Abajo a la derecha: Tamaño del poro de 1000 voxeles al tamaño máximo, este grupo esta formado por los 91 poros utilizados para evaluar la implementación de SPH. 90

6.31. Volúmenes de poros con sus estructuras de esqueleto internas, el esqueleto se utiliza para definir las áreas de sección transversal con las que se calcula el flujo volumétrico. 91

6.32. Resultado de simulación numérica para diferentes tipos de estructura de poros. La figura muestra un poro (arriba a la izquierda) con una estructura más similar a una estructura tubular, (arriba a la derecha) muestra una estructura de poro con una forma similar a un esferoide donde las condiciones de entrada y salida son más difíciles de definir. (Abajo) Estructura de poros con dos estructuras tubulares que se cruzan. 92

6.33. Volúmenes de poros con sus estructuras de esqueleto internas, el esqueleto se utiliza para definir las áreas de sección transversal con las que se calcula el flujo volumétrico. 93

6.34. (Arriba) Distribución de boxplot de la eficiencia de flujo de poros en escala logarítmica, el poro 7 tiene la mayor eficiencia (su forma es la mas tubular) y los poros 18 y 90 la menor (ambos amorfos). (Abajo) Distribución de Pareto, sólo 9 poros están por encima del 10 % de la eficiencia. 94

Índice de cuadros

6.1. Comparación de desempeño entre implementaciones en escenario A. . . 58

6.2. Comparación de desempeño entre implementaciones en escenario B. . . 58

6.3. Comparación de desempeño entre implementaciones en escenario C. . . 58

6.4. Uso de memoria en GPU por parte de las implementaciones LUT, Alternative, Recursive Y Opt. 62

6.5. Uso de memoria en GPU por parte de las implementaciones LUT, Alternative, Recursive Y Opt. 62

6.6. Resultados de desempeño de implementaciones OPT y SORTED empleando un Octree con nivel máximo de profundidad 7, usando los 3 escenarios de Exp 1. 65

6.7. Resultados de desempeño de implementaciones OPT y SORTED empleando un Octree con nivel máximo de profundidad 8, usando los 3 escenarios de Exp 1. 66

6.8. Resultados de desempeño de implementaciones OPT y SORTED empleando un Octree con nivel máximo de profundidad 8, usando los 3 escenarios de Exp 1. 67

6.9. Parámetros empleados para pruebas de flujo laminar en tubo usando ecuación de Poiseuille. 69

6.10. Tabla con resultados de desempeño obtenido para los procesos de simulación de fluido en escenarios de nubes de puntos aleatorias. Se utilizó la implementación Opt y un Octree con nivel de profundidad máxima 8. 77

6.11. Tabla con resultados de desempeño obtenido para los tres escenarios de simulación de fluido con objeto estático. 78

6.12. Tabla con resultados de desempeño obtenido para simulación con implementación de cálculo de vecindad SORTED. Escenarios A, B Y C. . . . 83

6.13. Tabla con resultados de desempeño obtenido para estudio de forma con implementación de cálculo de vecindad SORTED. 88

6.14. Numero promedio de partículas en nodos de Octree a diferentes niveles de profundidad máxima. 88

Índice general

1. Introducción **1**

1.1. Simulación de fluidos 3

1.2. Simulación de cuerpos deformables 4

1.3. Simuladores interactivos 6

1.3.1. Motores de videojuegos 6

1.3.2. Simuladores quirúrgicos 7

1.4. *Big Data* 8

1.5. Hipótesis y contribución 10

1.6. Objetivo 11

1.7. Plataforma de desarrollo 11

1.8. Publicaciones 12

2. Cálculo de vecindades **13**

2.1. Subdivisión espacial 14

2.1.1. Octree 14

2.2. Indexado de estructuras de datos 16

2.2.1. Clave Morton 16

2.3. Ordenamiento espacial 18

2.4. Método propuesto de cálculo de vecindades 22

2.4.1. Algoritmo 24

3. Simulación de fluidos **29**

3.1. Método de Hidrodinámica de partículas suavizadas (*Smoothed Particle Hydrodinamics* SPH) 29

3.1.1. Fundamentos de SPH 30

3.2. Ecuaciones de Navier-Stokes 32

3.3. Modelo empleado para simulación de fluidos 33

3.4. Kernel de suavizado (Smoothing Kernel) 36

3.5. Implementación de Simulación de fluidos en conjunto con Octree 37

4. Cuerpos suaves y colisiones **40**

4.1. Cuerpos deformables con SPH 40

4.2. Modelo viscoelástico 41

4.3. Colisión entre partículas 43

4.3.1. Método de detección de colisiones propuesto 44

5. Big Data y sistemas de partículas **48**

5.1. *Big Data* y SPH 49

6. Experimentos y Resultados **53**

6.1. Exp 1. Calculo de vecindad 53

6.2. Exp 2. Cálculo de vecindad con ordenamiento por clave Morton 64

6.3. Exp 3. Simulación de fluidos 68

6.4. Pruebas de velocidad bajo diferentes condiciones 75

6.4.1. Exp 4. Simulación con implementación Opt 75

6.4.2. Exp 5. Simulación de fluido con objeto estático 78

6.4.3. Exp 5a. Simulación de fluido con objeto deformable (Viscoelástico) 80

6.4.4. Exp 6. Simulación de fluidos (Sorted) 82

6.5. Exp 7. Estudio de forma con SPH. 87

6.6. Exp 8. Simulación de flujo en formas diversas (poros). 89

7. Conclusiones **95**

7.1. Discusión 97

7.2. Trabajo futuro 98

A. Repositorios de código **100**

B. Método de búsqueda de vecinos **101**

C. Estructuras de datos para experimento 1 **103**

D. Funciones para uso de claves Morton (CUDA) **106**

Capítulo 1

Introducción

La simulación de la realidad mediante modelos abstractos presenta un desafío para la computación dado que conforme más se entiende un fenómeno los modelos para su representación se vuelve más complejo y los métodos numéricos para su simulación requieren de más recursos computacionales o de métodos que aceleren los cálculos necesarios para simular los modelos. Por otro lado, en el caso de las simulaciones interactivas por computadora se hacen compromisos entre realismo, inmersión y presencia dependiendo de la aplicación para la que se construyen, pero de igual forma se busca mejorar la presentación visual de los fenómenos [1].

Hoy en día las simulaciones por computadora son ampliamente usadas, si se cuenta con un modelo (matemático, físico, químico, biológico, clínico, social, económico, etc.) del comportamiento de un fenómeno es posible aplicar o desarrollar un método numérico con el cual se puede traducir ese modelo en un programa de cómputo, el cual puede ejecutar una simulación usando dicho modelo para obtener información de interés sobre el fenómeno estudiado [2].

Un conjunto de soluciones populares para la simulación por computadora, son las que utilizan como primitiva de representación del volumen un elemento finito (por ejemplo un tetraedro), llamados por esto métodos de elemento finito, FEM por sus siglas en inglés [3], [4]. Otro tipo de métodos proponen eliminar el uso de una primitiva de representación para resolver el modelo y en su lugar proponen el uso de sistemas de partículas para la representación y solución de la simulación [5]-[7].

Una parte importante en el desarrollo de los motores de simulación son la serie de herramientas necesarias para la solución numérica por métodos computacionales de los modelos de una forma eficiente. Para ello es necesario desarrollar algoritmos de computadora que aprovechen las capacidades de cómputo del hardware utilizado.También es necesario considerar los recursos de cómputo disponibles para la ejecución de las diversas propuestas de motores de simulación, ya que pueden ser limitados por el tipo de recursos que el usuario final tendrá a su disposición y el tipo de dispositivo donde se planea ejecutar la solución.

Con el desarrollo del paradigma de programación en paralelo una forma de incidir en las posibilidades de simulación de un motor se basa en la creación de nuevos métodos que exploten las capacidades de sistemas de múltiples núcleos con los cuales se pueda mejorar la calidad de la solución sin incrementar los recursos de hardware. Y por otro lado que estos mismos métodos sean escalables para que si existe una mejora en las capacidades del hardware los mismos métodos hagan uso de esto para aumentar las capacidades de simulación del motor [8].

Conforme a las especificaciones de las simulaciones, se decide qué tipo de enfoque seguir. Un ejemplo, es un simulador interactivo, el cual debe enfocarse en dos aspectos fundamentales, inmersión y presencia. Siendo la inmersión una forma de describir como el usuario mediante sus sentidos percibe los eventos recreados virtualmente. Por otro lado, la presencia se refiere al modo en que el usuario interactúa con los elementos presentes en la simulación. Cuando cumplimos con estos aspectos podemos proporcionar una simulación interactiva con la cual el usuario puede interaccionar en tiempo real [9], este tipo de simulaciones pueden ser utilizadas como medios de entretenimiento, o herramientas de entrenamiento con la que se puede adquirir destrezas y habilidades.

Otro tipo de motores se basan en proveer soluciones precisas y exactas a diversos fenómenos para el estudio de estos o la aplicación en soluciones de ingeniería en diversas áreas. Estos motores al requerir mayor precisión necesitan una menor dimensión en las primitivas de representación en soluciones como FEM o un número mayor de partículas por área para métodos como SPH. Lo anterior hace que aumente los requisitos de equipo de cómputo necesarios para la simulación y los tiempos de cómputo para obtener la solución buscada. De la misma forma que para los motores de simulación interactivos es necesario desarrollar métodos que aceleren el tiempo de cómputo y aprovechen de la mejor manera posible los recursos de cómputo disponibles.

Al considerar las simulaciones libres de mallas, las cuales representen su dominio mediante sistemas de partículas, se tiene que una manera de mejorar el rendimiento es abordando un problema en común que compartan estos sistemas, este problema es el cálculo de vecindades. El problema de cálculo de vecindades es un caso especial del algoritmo del vecino más cercano, el cual tiene mejoras en su desempeño mediante la solución del problema del vendedor viajero, TSP por las siglas en inglés de *Travelling Salesman Problem* [10].

1.1. Simulación de fluidos

La simulación de fluidos se puede realizar mediante métodos basados en mallas o métodos libres de mallas, sin embargo, la naturaleza del comportamiento de los fluidos requiere que los métodos puedan manejar superficies de fluido libre, deformación de fronteras y cambios drásticos de forma. En vista de lo anterior los métodos libres de malla son la opción que mejor se adapta a esas condiciones.

En el análisis de Zhang et al. [11] se mencionan 4 razones principales para emplear métodos basados en hidrodinámica de partículas suavizadas, en inglés *Smoothed Particle Hydrodynamics* (SPH), para la simulación de fluidos. Las razones que se considera decisivas son su capacidad de representar adecuadamente superficies de fluido libre, deformación de fronteras y cambios de forma, además de que es aplicable en problemas de micro y macro escala, por último, se toma en cuenta que aún no alcanza su madurez por lo que su realismo, estabilidad y precisión pueden seguir mejorando.

Dentro de las simulaciones de flujos basadas en SPH que tienen la característica de ser interactivas y que se ejecutan en tiempo real, destaca el método de Fluidos basados en posición (*Position Based Fluids*) que fue presentado por Müller et al. [12], [13], Este método ha demostrado ser aplicable en la simulación de fluidos que interactúan con solidos deformables, por ejemplo, la interacción de sangre con estructuras vasculares [14], sin embargo, la única validación del realismo es visual. Este método goza de gran aceptación para aplicaciones interactivas y actualmente se conoce como física unificada de partículas [15].

Dos de los problemas a mejorar dentro de todas las diversas áreas de estudio de

los métodos libres de mallas son la estabilidad de las soluciones y la eficiencia de los métodos implementados. En el trabajo de Takahashi et al. [16] se propone el uso de diferentes kernels que mejoran la estabilidad y evitan errores durante la simulación. Y con respecto a la eficiencia existen trabajos como el de Chen et al. [17] donde los autores se enfocan en acelerar la búsqueda de vecinos mediante métodos de división espacial, con lo que se mejora el desempeño de la simulación.

Finalmente, existen propuestas que ayudan en ambos aspectos, mejoran la estabilidad de la solución y por como definen el problema también reducen en algunos casos el número de cálculos para ciertas partículas, uno de estos métodos es el SPH adaptativo (Adaptive SPH), este método es usado en el trabajo de He et al. [18] para reducir los la cantidad de partículas usadas durante la simulación, utilizando un muestreo dinámico de partículas, lo cual tiene como resultado partículas con diferente radio.

Los métodos antes mencionados tienen una característica en común, dada la naturaleza de las soluciones basadas en partículas suavizadas, se puede aprovechar el computo en paralelo para acelerar los cálculos y lograr simulaciones en tiempo real.

1.2. Simulación de cuerpos deformables

Siguiendo la clasificación de Zhang et al.[19], los modelos empleados para simular cuerpos deformables pueden ser clasificados como modelos heurísticos, modelos basados en mecánica del continuo y otros que utilizan diferentes aproximaciones para lograr el modelado de un cuerpo deformable.

Entre los modelos heurísticos existen los que se basan en la geometría de los volúmenes o las superficies para resolver la deformación de los objetos. Un ejemplo de este tipo de modelado es el metodo de la deformación libre de forma (en inglés Free-form deformation ,FFD)[20], en donde puntos de control y una interpolación de Bézier es usada para deformar los objetos. Otros clásicos metodos que han sido mejorados con el tiempo para lidiar con modelos mas complejos de superficies son por ejemplo la implementación del metodos de masas y resortes reportado por Duan [21]. Un modelo similar al modelo de masas y resortes, pero con un enfoque más simple, es el algoritmo ChainMail que presenta Frisken [22], el cual es muy eficiente pero poco realista.

Otro tipo de modelos heurísticos es el de dinámica basada en posición (PBD) [23], donde los objetos se simulan utilizando una serie de puntos y de reglas que dictaminan el comportamiento de estos.

Ninguno de los métodos presentados hace uso de las propiedades mecánicas de los objetos a modelar, la representación obtenida no es físicamente correcta pero su comportamiento visual es satisfactorio y tiene muchas aplicaciones por ejemplo en videojuegos o simulación tejidos en simuladores quirúrgicos.

Los métodos basados en mecánica del continuo tienen implementación basadas en métodos con mallas y libres de mallas.

Entre los metodos Eulerianos basados en mallas los más populares son las implementaciones basadas en el metodo de elemento finito (Finite Element Method, FEM), este método aproxima las leyes constitutivas en función de los elementos (basados en alguna primitivas de representación) que componen la malla con la que se construye el objeto. Las ecuaciones individuales del comportamiento para cada elemento son conjuntadas en un sistema de ecuaciones que modela el comportamiento mecánico de los objetos de interés. En este método se pueden integrar y medir propiedades de materiales deformables como lo es el módulo de Young y el radio de Poisson, esto permite que el método tenga una gran precisión y realismo [24], sin embargo, su costo computacional y complejidad es alta.

Existen simplificaciones a FEM que reducen el tiempo computacional para poder para cierto tamaño de objetos dar una solución en tiempo real [25], esto con un costo en el realismo y precisión de la simulación.

Los métodos libres de mallas son métodos de formulación lagrangiana los cuales no emplean primitivas de representación malladas para definir las ecuaciones que modelan el comportamiento del sistema, estos métodos representan el dominio mediante el uso de partículas las cuales se distribuyen de manera arbitraria dentro de este dominio. Ejemplos de estos métodos son *Smoothed Particle Hydrodinamics* (SPH) y *Point-Colocation-based Method of Finite Spheres* (PCMFS) [26], [27]. Estos métodos emplean cálculos basados en el conjunto de partículas dentro de una vecindad mediante la aplicación de ponderaciones mediante funciones de suavizado (kernel). Ejemplos donde se simulan cuerpos deformables mediante SPH son el presentado por Desbrum

et al. [28] y la simulación de tejidos suaves propuesta por Palyanov et al. [29].

Dentro de los modelos que no entran en los dos primeros tipos de clasificación se encuentran los que se basan en alguna otra forma para determinar cómo se deformaría un cuerpo ante una fuerza, entre estos se encuentran por ejemplo los que emplean redes neuronales como el trabajo presentado por Zhang et al. [30] donde se simulan tejidos suaves para simulaciones quirúrgicas.

1.3. Simuladores interactivos

El realismo en la simulación depende en gran medida de la caracterización que se desean modelar del objeto de interés, las cuales dependen de los materiales que lo conforman, por ejemplo, un tejido bajo condiciones *in vivo* requiere si se busca tener un alto realismo modelos viscoelásticos del tejido los cuales sin son muy complejos pueden afectar el tiempo computacional y no ser interactivos. Así en el caso de la simulación de objetos interactivos se ocupa analizar la relación entre representación realista, efecto visual y apreciación del usuario.

Un tipo de interactividad es alcanzada cuando una simulación se ejecuta en tiempo real, este desafío lleva a buscar modelos matemáticos más eficientes, reducir su complejidad y maximizar el aprovechamiento del poder de cómputo, ejemplo del último punto son el cómputo en paralelo, calculo previo de datos y uso de bases de datos.

1.3.1. Motores de videojuegos

Un ejemplo de aplicaciones de simuladores que requieren la ejecución en tiempo real son los motores de videojuegos, estos requieren que el usuario pueda mandar entradas por lo general a través de un control o teclado las cuales tienen que ser usadas por el motor para interactuar en tiempo real con el mundo virtual.

Ejemplos populares de estos motores son UNREAL [31] y UNITY [32], donde se puede observar que para llevar a cabo la simulación de una escena interactiva existen varios puntos a resolver por el motor. Respecto a las áreas que se relacionan con los temas presentados en este trabajo están la simulación de partículas en CPU y GPU, la

solución de movimiento predeterminado, la simulación de AI por métodos de árboles, el modelado de objetos plásticos destruibles, simulación y modelado de fenómenos de la luz y modelado de materiales por medio de métodos de sombreado.

Todos estos métodos se basan en ecuaciones e implementaciones algorítmicas que aproximan a la realidad, limitadas todas en que la solución encontrada sea posible ejecutarla en tiempo real (30, 60 o 120 cuadros por segundo) y es por esto por lo que se presentan limitaciones comunes en las implementaciones para videojuegos, por ejemplo:

- Simulación de tejidos y objetos viscoelásticos por medio de modelos elásticos.
- Objetos destruibles limitados a modelos plásticos.
- Modelado de la luz por métodos algebraicos basados en mallas [33].
- AI basado en arboles de decisiones que buscan simular la complejidad a través de una serie de decisiones expresadas por ramas interconectadas.
- Sistemas de partículas con bajo numero de elementos enfocados a simulaciones locales o de dimensiones pequeñas.
- Limitada interactividad entre objetos simulados físicamente, cuerpos deformables e instrumentos manejados por el usuario. Estos pueden mejorar con desarrollos que mejoran las bases del motor de videojuegos para resolver estas interacciones, por ejemplo, los simuladores de vuelo que han sido relanzados recientemente [34].

1.3.2. Simuladores quirúrgicos

Los simuladores quirúrgicos tienen como finalidad la adquisición de habilidades mediante la interacción de un usuario con modelos virtuales que replican el comportamiento físico de tejidos biológicos. En la actualidad el entrenamiento de cirujanos es *in vivo*, esto limita las oportunidades de entrenamiento y la variedad de casos en los que se puede entrenar. Estos problemas pueden evitarse mediante el uso de simuladores quirúrgicos como herramientas didácticas, por lo que es posible incrementar el tiempo y disponibilidad de oportunidades de entrenamiento sin incurrir en una inversión económica mayor, como lo es el costo asociado a un quirófano [35].

Al considerar las características, necesidades y metas de los simuladores quirúrgicos es necesario construir herramientas que sean capaces de visualizar datos (ambientes de realidad virtual) obtenidos mediante modelos que se apeguen al comportamiento real de los diferentes actores que intervienen en la simulación, al mismo tiempo que se permita a un usuario la interacción en tiempo real y sea posible la obtención de datos que permitan evaluar dicha interacción.

Los simuladores quirúrgicos requieren de métodos de simulación que permitan obtención de datos que permitan realizar un despliegue gráfico de entre 30Hz a 60Hz, mientras que para la retroalimentación táctil o háptica se requiere retroalimentación de fuerza con una frecuencia de interacción de 1kHz [36], [37], esta característica hace necesario que el método para simular pueda resolverse en espacios determinados de tiempo sin sacrificar el realismo en el comportamiento de los objetos simulados. En el trabajo de Zhang se menciona que, algunos de los desafíos que deben vencer los simuladores quirúrgicos se presentan en la forma de simulación realista y simulación en tiempo real [19].

Algunos ejemplos de estos sistemas son los trabajos presentados en [35], [38]-[40], donde se puede observar que estos simuladores deben de cumplir todos los puntos discutidos para poder ser implementados como entrenadores quirúrgicos.

1.4. *Big Data*

Las implantaciones de soluciones de sistema de partículas suavizadas que serán utilizadas para modelar fenómenos con el fin de obtener mediciones requieren métodos que permita el análisis de los resultados obtenidos. En los métodos basados en mallas estas son utilizadas para la presentación y el análisis de los resultados, así los métodos Eulerianos tienen en si mismos la estructura para presentar sus resultados.

En la aproximación Lagrangiana de las partículas suavizadas es común observar implementaciones [41] que muestran los resultados a partir del valor de cada partícula como mapa de colores 1.1 presentando con esto una representación del comportamiento. Por otro lado, es necesario también desarrollar métodos que permitan hacer minería de datos (*Data Mining*) en las soluciones obtenidas donde se presenta el problema de

la cantidad de información que debe de ser analizada lo cual hace necesario explorar implementaciones de *Big Data* para el análisis de la solución. Donde la minería de datos no solo permite estudiar los resultados de la simulación, sino que también permite generar modelos estadísticos junto a otras variables que permiten modelar fenómenos complejos y generar métodos para la detección de patrones [42], [43].

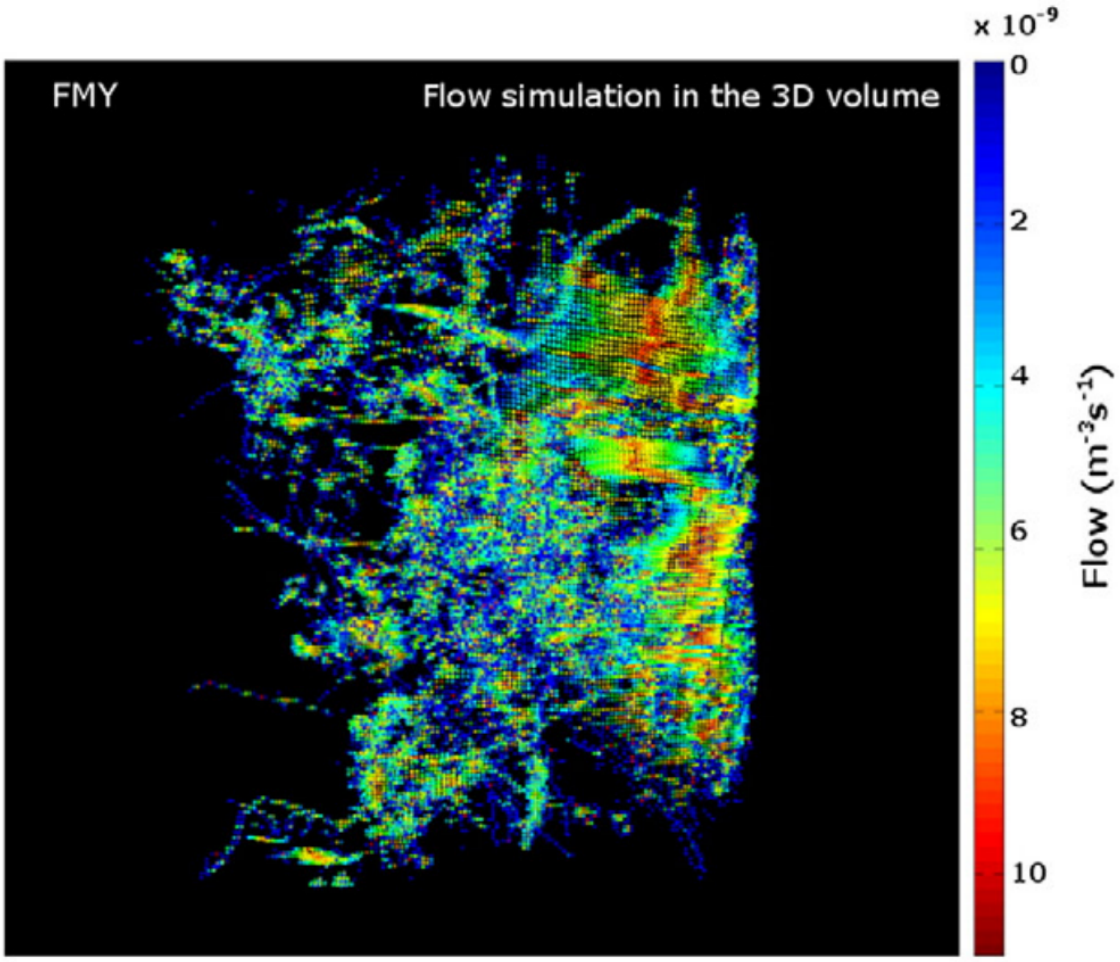

Figura 1.1: Imagen tomada de Dal Ferro et al. [41]. Mapa de color de estructura conductora de agua en subsuelo.

En un artículo realizado durante la elaboración de este trabajo, se encontró qué en el área médica, existen problemas que requieren de la implementación de técnicas de *Big Data* para su solución [44]. En dicho trabajo se describe que es posible aplicar técnicas de aprendizaje de máquina para realizar análisis de este tipo de información. Esto a su vez, muestra que el uso de recursos de cómputo como las unidades de procesamiento gráfico, permiten, mediante un enfoque diferente, analizar grandes cantidades

de información.

Otras dos implementaciones presentadas en este trabajo donde se aplica los métodos desarrollados en este trabajo para el análisis de fenómenos y objetos, una de estas aplicaciones es para la obtención de descriptores en superficies de objetos arqueológicos [45] y otra es para la obtención de medidas especificas del comportamiento del flujo dentro de micro-poros utilizando una representación euleriana que hace uso del Octree para presentar de una forma mas clara los resultados del método de SPH y permitir el análisis estadístico de los resultados.

1.5. Hipótesis y contribución

Los métodos libres de mallas representan el dominio mediante sistemas de partículas, la resolución con la que se puede representar el dominio depende de la distancia entre partículas y el volumen que cada una de ellas representa. La solución numérica de estos métodos se hace a partir de la vecindad de cada una de las partículas del sistema. Por lo tanto, un incremento en la resolución y precisión de la simulación requiere un incremento del numero de partículas y con esto aumenta el número de cálculos para obtener la vecindad de cada una de estas.

En este trabajo se propone como solución al problema de un alto numero de partículas y de los cálculos necesarios para definir la vecindad de las partículas de forma dinámica, un método de calculo de vecindades para sistemas de partículas. Este método emplea indexación por claves Morton, ordenamiento y subdivisión espacial con un Octree dinámico, el cual saca provecho de la capacidad de cómputo de en una arquitectura en paralelo.

En este aspecto, las arquitecturas en paralelo que emplean unidades de procesamiento gráficas tienen sus propias ventajas y limitaciones sobre los procesadores tradicionales, por lo que los métodos tradicionales no garantizan su aprovechamiento de manera eficiente. Para resolver este problema el método presentado de Octree dinámico hace uso de una serie de ecuaciones altamente paralelizables con una reducción en los accesos a memoria con lo que se busca obtener un método óptimo para calculo de vecindades en unidades de procesamiento gráficas.

1.6. Objetivo

Desarrollar un método para mejorar el desempeño en los cálculos requeridos para simulaciones libres de mallas, el cual aproveche las capacidades de cómputo que proporcionan las arquitecturas en paralelo que implementan las unidades de procesamiento gráfico.

Modificar la forma de crear y navegar un Octree para hacer más eficientes los cálculos de vecindades en sistemas de partículas en la implementación en paralelo.

Aplicar el método propuesto para verificar la compatibilidad con una simulación libre de mallas (SPH), la cual pueda simular flujo y cuerpos deformables.

Explorar las capacidades del método propuesto en aplicaciones basadas en sistemas de partículas, donde no necesariamente sean simulaciones, pero que se beneficien del poder de cómputo de las arquitecturas en paralelo proporcionadas por unidades de procesamiento gráfico y el cálculo de valores en superficies o volúmenes definidos por nubes de puntos.

1.7. Plataforma de desarrollo

A continuación, se enumeran las herramientas de desarrollo empleadas para el desarrollo e implementación de los métodos propuestos en esta tesis:

- Programación orientada a objetos en C++.
- Herramienta de desarrollo Visual Studio [46] (2013, 2015, 2017, 2019).
- Sistema Operativo Windows.
- Arquitectura de cómputo en paralelo CUDA [47].
- Despliegue gráfico mediante la biblioteca OpenGL.

La arquitectura de cómputo en paralelo CUDA (Compute Unified Device Architechture) es una plataforma de computo en paralelo y modelo de interfaz de programación

de aplicaciones desarrollada por la empresa Nvidia. Esta plataforma permite el desarrollo cómputo de propósito general para ser ejecutado en unidades de procesamiento gráfico.

1.8. Publicaciones

Producto de este trabajo se realizaron las siguientes publicaciones:

Artículos publicados:

- Soriano-Valdez, D., Pelaez-Ballestas, I., Manrique de Lara, A. et al. The basics of data, big data, and machine learning in clinical practice. Clin Rheumatol 40, 11–23 (2021). https://doi.org/10.1007/s10067-020-05196-z.

Congresos:

- Valdez D.A.S. et al. (2020) CUDA Implementation of a Point Cloud Shape Descriptor Method for Archaeological Studies. In: Blanc-Talon J., Delmas P., Philips W., Popescu D., Scheunders P. (eds) Advanced Concepts for Intelligent Vision Systems. ACIVS 2020. Lecture Notes in Computer Science, vol 12002. Springer, Cham. https://doi.org/10.1007/978-3-030-40605-9_39.

Se contribuyó con el desarrollo de un modulo de simulación con GPU a SMAS de Biocomlab, http://www.biocomlab.com/apps/SMAS.html .

Capítulo 2

Cálculo de vecindades en nubes de partículas

Los métodos de simulación libres de mallas representan el dominio mediante un conjunto de partículas las cuales se distribuyen de manera arbitraria, estas partículas no se vinculan entre sí mediante el uso de algún tipo de elemento geométrico, debido a esta característica este tipo de métodos requieren de conocer el conjunto de vecinos más cercanos a cada partícula a fin de realizar los cálculos necesarios para ejecutar la simulación.

La búsqueda de los vecinos más cercanos para un conjunto de n partículas empleando un método de fuerza bruta tiene una complejidad de $O(n^2)$, donde se observa que esta forma de búsqueda en los métodos de partículas suavizadas tendrá un impacto alto, empeorando esto cuando se considera que la mejor solución a un problema necesita que se represente el domino con tantas partículas como sea posible.

Algunos de los enfoques más reportados en la literatura se basan en métodos de división y ordenamiento espaciales [48]-[51].

El método que se propone en este trabajo está basado en división espacial mediante el uso de una estructura de datos conocida como árbol octal (Octree), la cual es indexada mediante un valor conocido como clave Morton, MK por su nombre en inglés, Morton Key [52].

2.1. Subdivisión espacial

Las estrategias de subdivisión espacial mas utilizadas son las de rejillas (Grid) y la de árboles. Tradicionalmente se trabaja con tres tipos de árboles, binarios (Binary tree), cuaternarios (Quad tree), octales (Octree), pero es posible definir arboles en cualquier dimensión. En este trabajo se hace uso de una estructura de datos de tipo Octree.

2.1.1. Octree

En el método propuesto, la división del espacio de trabajo de 3 dimensiones $R^3(X, Y, Z)$ se realiza de forma simétrica en cada una de las direcciones de los ejes de referencia, con esto tenemos como resultado una división inicial en 8 subsecciones, cada una de estas subsecciones recibe el nombre de nodo N. Esta división se puede aplicar de manera recursiva sobre cada nodo N, adicional a esto definimos al espacio de trabajo en su totalidad como nodo raíz N_{ROOT}.

Los nodos N se encuentran contenidos a un cierto nivel de profundidad $[0, 10]\forall\mathbb{N}$, al dividir en 8 subsecciones cada uno de los nodos N el nodo que fue dividido inicialmente se convierte en nodo padre N_{parent} y los nodos resultantes de la división se convierten en nodos hijo N_{child}. El nivel de profundidad se denota mediante la relación $DL_O(N)$ y se tiene las siguientes condiciones iniciales $DL_O(N_{ROOT}) = 0$ y $DL_O(N_{parent}) = DL_O(N_{child}) - 1$. La relación entre los nodos hijo N_{child} y su nodo padre N_{parent} tiene las siguientes propiedades $N_{parent} = \bigcup_i N^i_{child}$ y $N^j \cap N^k = \emptyset$ siempre que se cumpla la condición $DL_O(N^j) = DL_O(N^k)$. En la Figura 2.1 se muestra la división del espacio conforme se incrementa el nivel de profundidad del Octree, mientras que en la Figura 2.2 se muestra la relación jerárquica entre nodos padre y nodos hijo.

El enfoque usual basado en división espacial mediante el uso de un Octree busca tener un elemento en cada uno de los nodos, pero para lograr eso con nuestro enfoque, es necesario que el nivel de profundidad del Octree no esté limitado. Dado que nosotros limitamos el nivel de profundidad, ya que indexamos mediante una MK, la cual tiene un tamaño finito, es que no podemos aplicar dicho enfoque y cambiamos el enfoque a solo contener un subconjunto de elementos en cada uno de los nodos a fin de reducir la

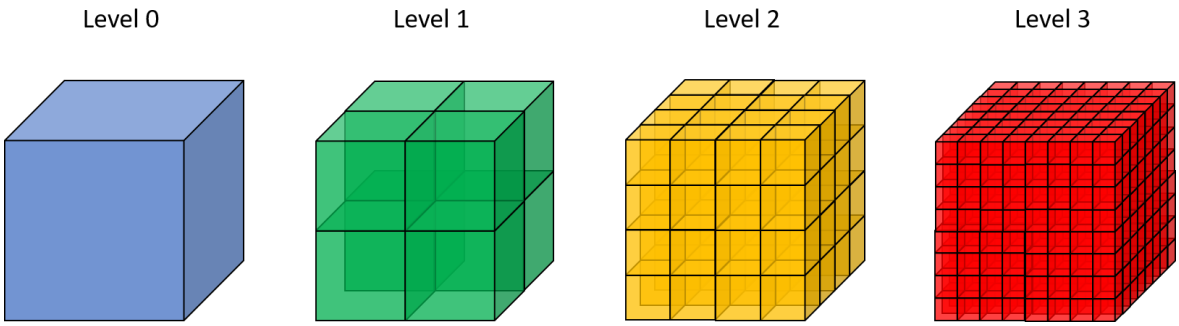

Figura 2.1: División espacial mediante el uso de nodos contenidos en un Octree a diferentes niveles de profundidad.

cantidad de cálculos necesarios para realizar la búsqueda de los vecinos más cercanos.

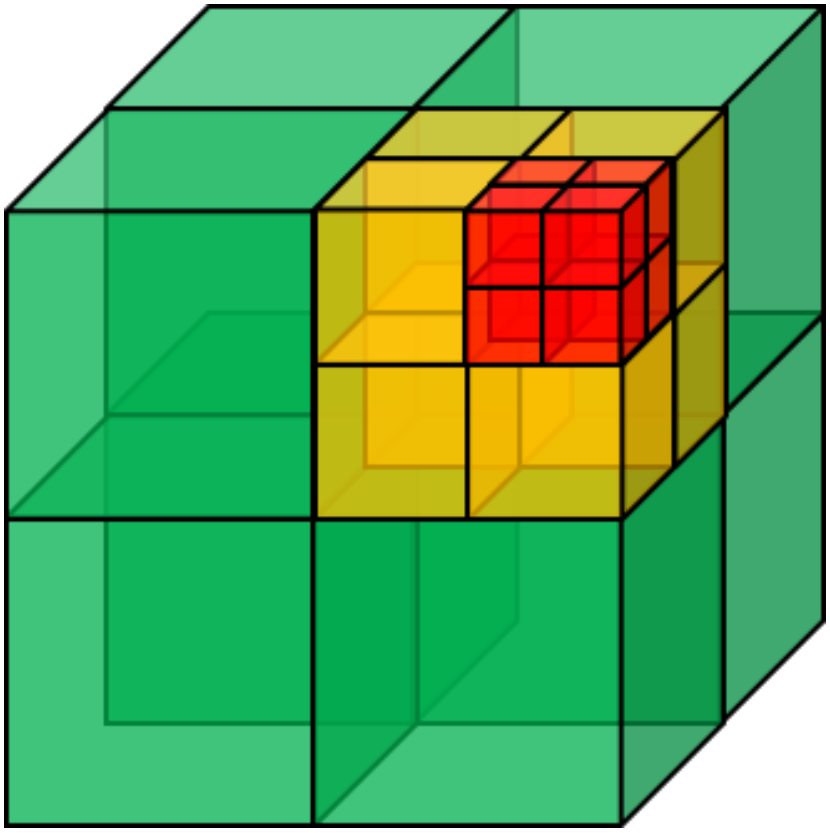

Figura 2.2: Relación entre nodos padre y nodos hijo en el Octree.

2.2. Indexado de estructuras de datos

El indexado de los nodos que componen el Octree nos permite usar reglas para navegar la estructura de datos sin la necesidad de emplear apuntadores, lo cual tiene como principal beneficio tener acceso aleatorio en lugar de secuencial. En este trabajo empleamos indexado mediante claves Morton, este proceso no es trivial por la geometría que describe la curva de indexado, pero tiene ventajas de acceso a elemento multidimensionales usando un único dato.

2.2.1. Clave Morton

La Clave Morton (MK) que se emplea para indexar los nodos se representa como un valor binario de 32 bits, donde los 2 bits más significativos corresponden a la etiqueta 01_2 o 00_2 para indicar si el nodo tiene partículas en su interior o está vacío respectivamente. Los 30 bits menos significativos son usados para almacenar el índice del nodo N.

La codificación de este índice requiere de 3 bits por cada nivel de profundidad en el que se encuentre el nodo N dentro del Octree, es por esto por lo que el nivel máximo con una clave de 3s bits es de 10. Cada una de estas triadas de bits es concatenada partiendo del nivel 1 hasta el nivel 10, esta concatenación se realiza desde los bits más significativos a los bits menos significativos. En la Figura 2.3 se muestra un esquema con la descripción de la MK usada.

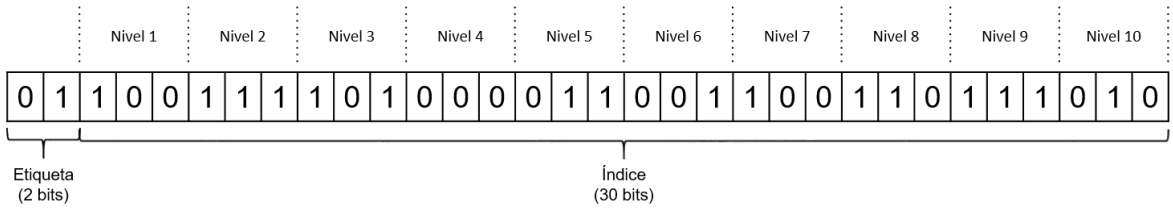

Figura 2.3: Diagrama de trama binaria que compone la clave Morton (MK).

El llenado del árbol inicia en los nodos en la máxima profundidad del Octree, se calcula el valor de MK para cada partícula y se asigna a el nodo correspondiente. Aunque la asignación inicial se hace en los nodos en la máxima profundidad, es posible conocer las partículas contenidas en un nodo con menos nivel de profundidad mediante

la relación de N_{parent} con N_{child}.

La construcción de la MK requiere calcular la posición normalizada pos_{Norm} de cada partícula, está normalización se genera empleando las dimensiones de Octree y el rango de valores que es posible obtener usando 10 bits, tomando como valor mínimo 0000000000_2 y valor máximo 1111111111_2. La MK se obtiene concatenando de manera secuencial los bits de los valores de posición normalizada $pos_{Norm}.x, pos_{Norm}.y, pos_{Norm}.z$ en triadas de la forma $bit_z bit_y bit_x$, la secuencia inicia con los bits más significativos a los menos significativos. La primer triada generada se coloca en los 3 bits más significativos de los 30 bits reservados para almacenar el índice y las subsecuentes en los siguientes 3 hasta terminar con los todos los bits disponibles en $pos_{Norm}.x, pos_{Norm}.y, pos_{Norm}.z$. Las triadas generadas corresponden a la posición de un N_{child} contenido dentro de un N_{parent} a cierto nivel de profundidad. En la Figura 2.4 se muestra un diagrama donde se ilustra el proceso de construcción de la MK.

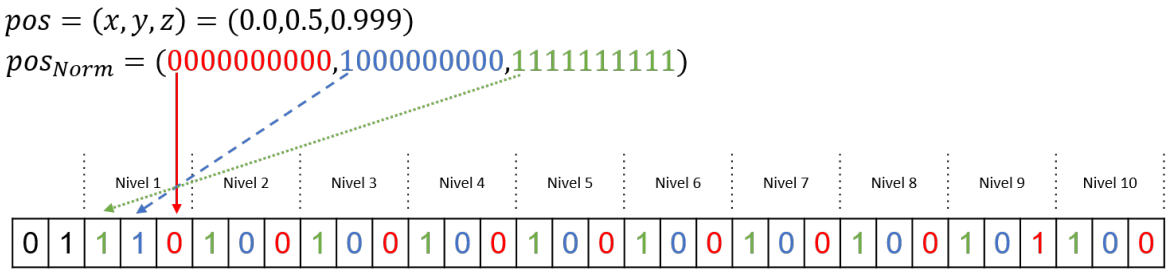

Figura 2.4: Construcción de clave Morton (MK) usando entrelazado binario a partir de posición.

Es posible usar solo una porción del índice de la MK, este depende del nivel de profundidad en el que estemos interesados trabajar, a esta porción le llamamos $subMK$. La $subMK$ también está compuesta de 32 bits, sin embargo, las triadas de los bits menos significativos siempre son 0. Por ejemplo, si se requiere la $subMk$ a nivel de profundidad 7 de la MK generada a partir de la posición pos de cualquier partícula p, se tendrá que las 3 triadas de bits menos significativos de la MK se convierten en 0, estas triadas corresponden a los niveles 10, 9 y 8.

Dado que ciertas MK y $subMK$ pueden tener la misma secuencia binaria, es necesario especificar el nivel de profundidad cuando se usan dichos valores.

Para usar la MK como un índice es necesario tener en cuenta las siguientes condiciones. Se considera que el Octree usado es un árbol completo, por lo que cada nodo

tiene 8 hijos en todos los niveles con excepción de los nodos de nivel máximo (los cuales no tienen nodos hijo). Si estas condiciones se cumplen es posible emplear la ecuación (2.1) para obtener el índice de un nodo a partir de su MK N_{MK},

$$Index = (\sum_{i=0}^{DL_O(N)-1} 8^i) + Val(N_{MK}) \tag{2.1}$$

Donde $Val(N_{MK})$ es el valor binario de los n bits de la MK con $n = (3)(DL_O(N))$, estos bits se toman partiendo de las triadas más significativas y se ignoran los 2 bits que son usados para etiquetado. Con la ecuación (2.2) es posible obtener la MK de un nodo N_{MK} a partir de un índice, solo se tiene que considerar que la ecuación no proporciona los 2 bits usados para etiquetado.

$$MK = Index - (\sum_{i=0}^{\log_8(Index)} 8^i)) \tag{2.2}$$

El indexado empleando MK da resultado a un ordenamiento conocido como *z-order* debido a la secuencia con que se organizan los elementos, el método propuesto aprovecha este tipo de ordenamiento para determinar los elementos que se encuentran en la vecindad mediante operaciones binarias. En la Figura 2.5 se muestra como el ordenamiento de los nodos al indexar mediante MK resulta en una secuencia que asemeja una z, de ahí el nombre de z-order.

2.3. Ordenamiento espacial

El algoritmo de búsqueda de vecinos es una generalización del algoritmo de búsqueda del vecino más cercano, dicho algoritmo ha sido empelado en la solución de diversos problemas, entre ellos el problema del vendedor viajero (TSP por su nombre en inglés Travelling Salesman Problem).

El problema del vendedor viajero se puede describir de manera general como un problema para determinar la ruta más corta de entre un conjunto de posibles rutas. Este problema ha sido resuelto con éxito mediante teoría de gráficas, pero también

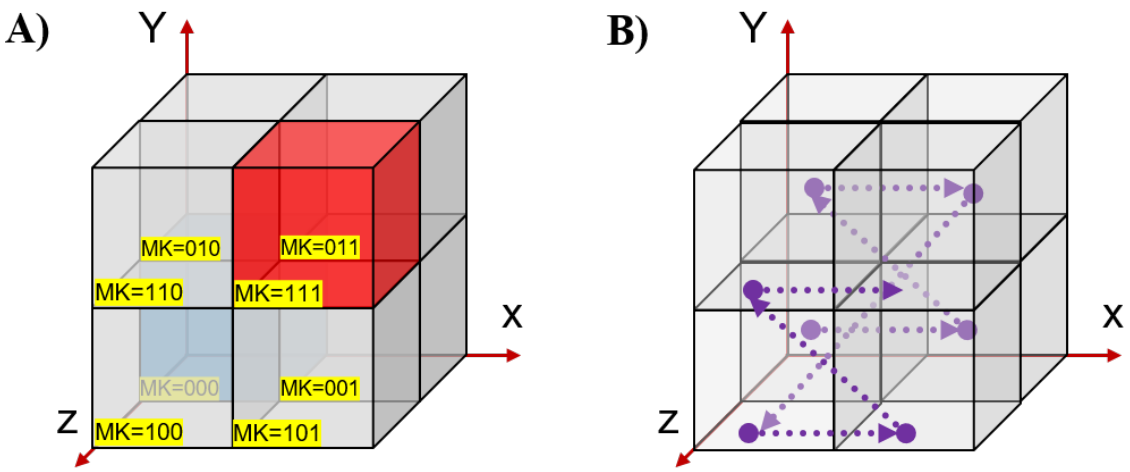

Figura 2.5: A) Clave Morton parcial (subMK) para nodos hijo dentro de un nodo padre. B) Secuencia de indexado empleando la curva z-order en 3 dimensiones.

mediante la búsqueda del vecino más cercano y empleando un enfoque codicioso o greedy en inglés; dicho enfoque tiene una complejidad $O(n^2)$.

En la bibliografía existen trabajos que buscan reducir la complejidad del algoritmo, como el de Bartholdi [10], donde logra mejorar la complejidad a $O(n \log n)$ mediante la inclusión de ordenamiento al conjunto de puntos a los que se mueve el vendedor. Siguiendo esta idea es que en este trabajo también se agrega un proceso de ordenamiento el método propuesto para la búsqueda de vecinos.

El método desarrollado para el cálculo de vecindades radiales hace uso de métodos para acelerar los algoritmos de búsqueda, estos son indexado mediante claves Morton (MK) y división espacial mediante el uso de árboles (Octree), sin embargo, este método requiere una gran cantidad de memoria, debido a que la implementación crea listas de elementos para cada una de las regiones en las que se divide el espacio. Cuando se consideran las limitaciones que las tarjetas gráficas tienen en memoria, así como las capacidades de cómputo que ofrecen las arquitecturas en paralelo, como lo es CUDA, es necesario considerar optimizaciones para el método que proponemos, las cuales aprovechen el poder de cómputo y sean eficientes en el uso de memoria en la GPU.

Para poder mitigar el problema de uso de memoria en GPU, se agrega un cambio en el método propuesto para el cálculo de vecindades radiales, este cambio agrega una etapa adicional en la que se realiza el ordenamiento de los elementos mediante el uso de

su MK. En la Figura 2.6 se ilustra el cambio al método y en qué punto se efectúa dicho cambio. Esto tiene un incremento en el tiempo de procesamiento ya que se sustituye una única etapa de asignación por dos etapas, una de ordenamiento usando MK y una de asignación de apuntadores.

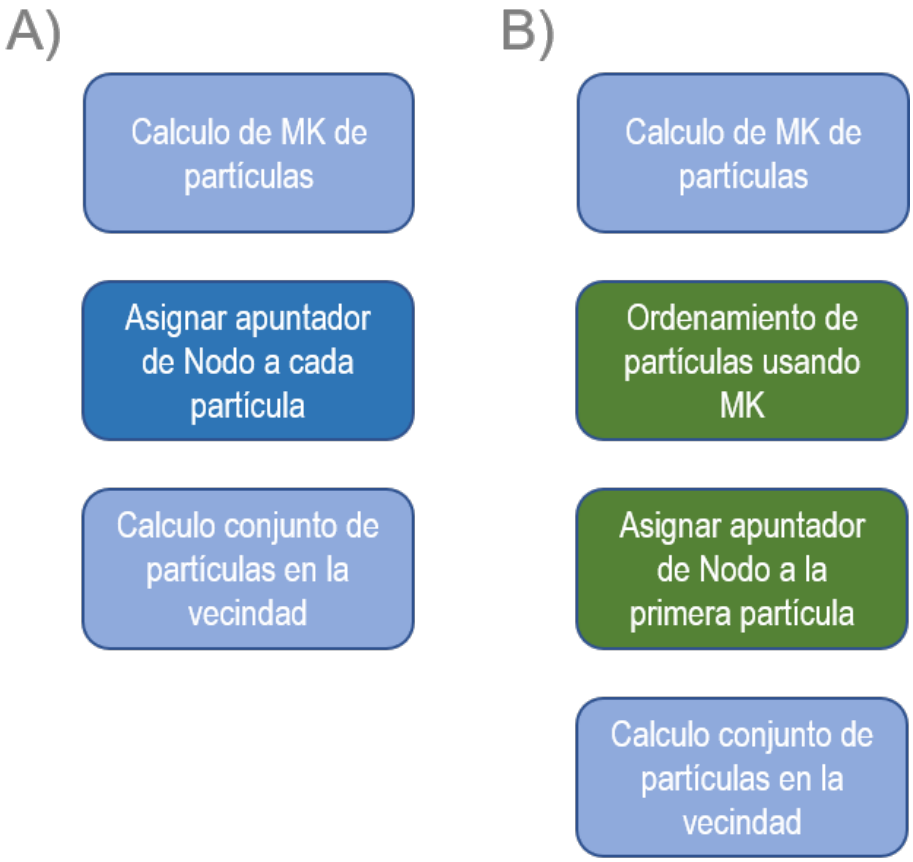

Figura 2.6: A) Etapas de cálculo de vecindades sin ordenamiento. B) Etapas de cálculo de vecindades empleando ordenamiento por MK.

Dado que no es objetivo de este trabajo el abarcar los diferentes tipos de ordenamientos que pueden usarse en arquitecturas en paralelo, es que el proceso de ordenamiento se realiza empleando una biblioteca llamada Thrust. Esta biblioteca proporciona algoritmos en paralelo para realizar ordenamientos de vectores, además de que permite aprovechar el poder de cómputo de la arquitectura CUDA de Nvidia. Uno de los métodos que implementa esta biblioteca para ordenar conjuntos es el ordenamiento Radix, Radix Sort en idioma inglés, dicha implementación emplea el rediseño de Satish et al. [53], con lo que saca provecho de las capacidades de cómputo de las GPU; para mayor detalle se recomienda consultar lo escrito por Bell y Hoberock [54].

Empleando la biblioteca Thrust se realiza el ordenamiento mediante claves Morton (MK), considerando el nivel de profundidad deseado de las partículas. Esto nos da como resultado un conjunto ordenado, donde podemos garantizar que usando solo un apuntador a una partícula podemos obtener todas las partículas que corresponden a un nodo del Octree. Para lograr esto usamos la primera partícula que encontramos y asignamos a un apuntador el nodo con la misma MK, posteriormente contamos el número de partículas que comparte dicha MK, esto es simple ya que la MK de las partículas solo cambia cuando se trata de partículas en otro nodo, por lo que al encontrar una MK diferente, repetimos la asignación y volvemos a contar hasta que terminamos con las partículas.

Lo descrito anteriormente permite hacer más eficiente la asignación de apuntadores, ya que, en lugar de tener un apuntador por cada partícula, es posible tener un apuntador por cada conjunto de partículas que comparten la misma clave Morton (MK). La implementación es simple y transparente, sin embargo, requiere agregar algunas operaciones atómicas al momento de paralelizar el proceso, de lo contrario el proceso es secuencial, lo cual se debe evitar para sacar provecho de la arquitectura en paralelo. En la Figura 2.7 se muestra el diagrama de memoria y apuntadores necesarios para vincular partículas y los nodos de la estructura del Octree si no se ordenan las partículas por clave Morton.

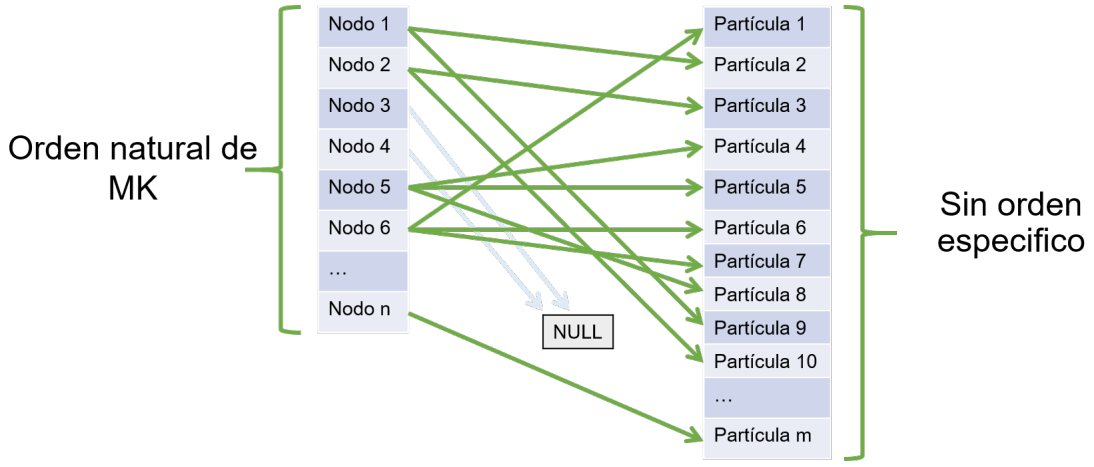

Figura 2.7: Diagrama de apuntadores necesarios en nodos para enlazar con el sistema de partículas sin un ordenamiento especifico.

Por otro lado, en la Figura 2.8 se muestra el mismo diagrama considerando el ordenamiento de partículas usando la calve Morton. Es evidente que el ordenamiento

reduce la cantidad de apuntadores requeridos, con lo que es más eficiente el uso de memoria y se reduce la complejidad de la estructura de datos empleada.

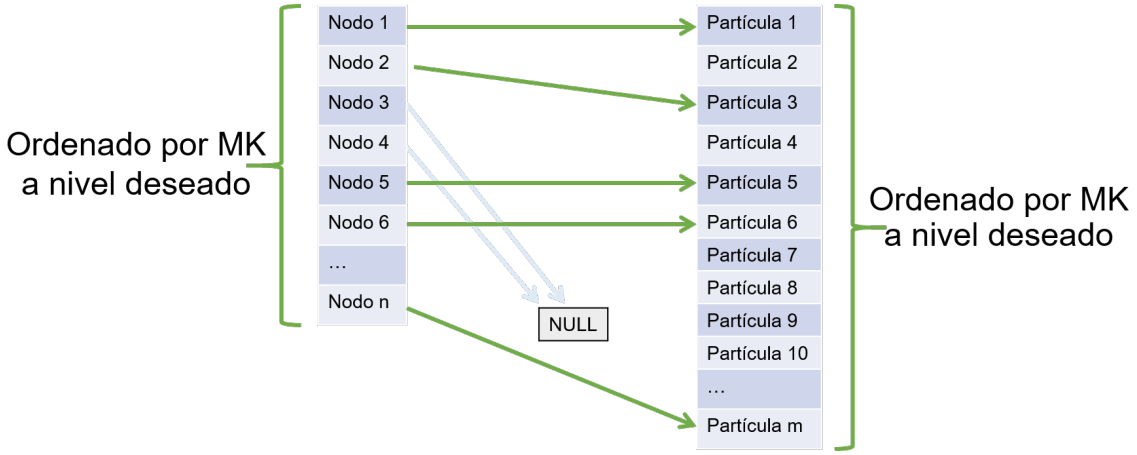

Figura 2.8: Diagrama de apuntadores necesarios en nodos para enlazar con el sistema de partículas ordenando las partículas mediante su clave Morton (MK).

2.4. Método propuesto de cálculo de vecindades

El método propuesto para el cálculo de vecindad radial emplea división espacial, con esto se reduce el conjunto de elementos contra los que se tiene que validar la vecindad radial, si además proponemos que el radio de búsqueda este contenido dentro del conjunto de nodos N que se encuentran a un cierto nivel de profundidad se puede obtener el conjunto de elementos que se encuentran dentro de la vecindad radial sin realizar ningún cálculo de distancia euclidiana.

Para poder determinar el conjunto de partículas que se encuentran dentro de la vecindad radial de una partícula p es necesario conocer su MK, al conocer su MK podemos determinar en qué nodo N se encuentra dentro del Octree y empleando este nodo como punto inicial de búsqueda $N = N_{base}$ podemos definir un conjunto de 26 nodos vecinos que son los más cercanos. Estos vecinos los definimos en tres subconjuntos que tienen la característica de ser adyacentes por cara, arista y vértice.

Vecino por cara:
$\{N_R, N_L, N_U, N_D, N_F, N_B\}$

Vecino por arista:
$\{N_{RU}, N_{RD}, N_{RF}, N_{RB}, N_{LU}, N_{LD}, N_{LF}, N_{LB}, N_{UF}, N_{UB}, N_{DF}, N_{DB}\}$
Vecinos por vértice:
$\{N_{RUF}, N_{RUB}, N_{RDF}, N_{RDB}, N_{LUF}, N_{LUB}, N_{LDF}, N_{LDB}\}$

En la Figura 2.9 se ilustra la vecindad para un nodo de un Octree con vecinos en todas sus direcciones.

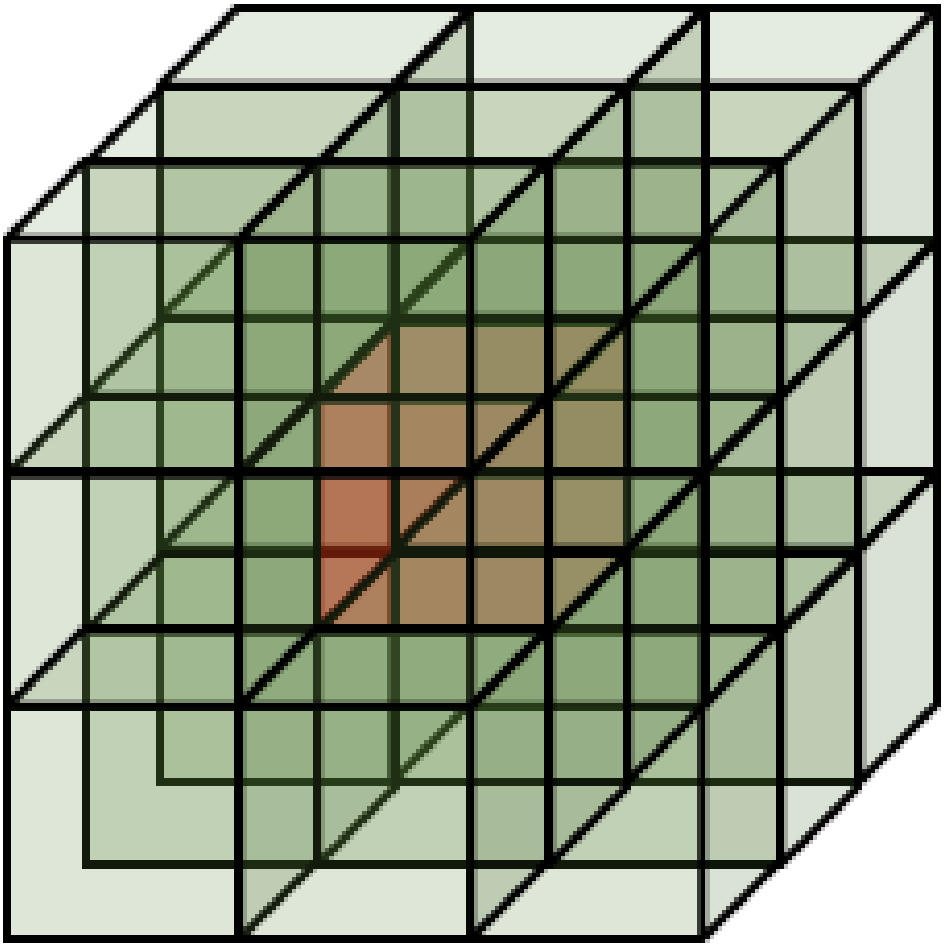

Figura 2.9: Vecindad radial de un nodo (naranja). Los nodos vecinos (verde) son todos los nodos adyacentes por cara, arista o vértice.

La vecindad de nodos propuestas tiene la característica de ser simétrica, por lo que se puede reducir la cardinalidad del conjunto de 26 a 13, dando como resultado que el conjunto de nodos donde se encuentran los vecinos consta de 14 nodos, los cuales son:

$\{N_{BASE}, N_R, N_U, N_F, N_{RU}, N_{RD}, N_{RF}, N_{RB}, N_{UF}, N_{DF}, N_{RUF}, N_{RUB}, N_{RDF}, N_{RDB}\}$

Aunque geométricamente estos nodos son fácilmente identificables a simple vista, en el Octree es difícil saber que vecino se encuentra en cada una de las posiciones requeridas, pero gracias al indexado mediante la MK es posible conocer estos nodos. A continuación, se muestra el algoritmo que se propone para encontrar vecinos por cara mediante el uso de MK. En la Figura 2.10 se muestran los subconjuntos de nodos para el cálculo de la vecindad propuesta, la cual consta de 14 nodos.

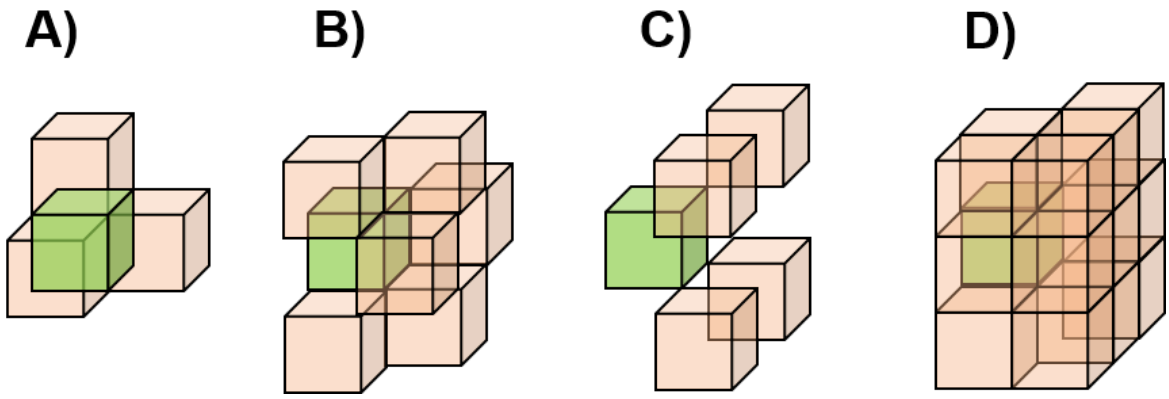

Figura 2.10: Conjuntos de nodos que conforman la vecindad empleada por el método. A) Vecinos por cara. B) Vecinos por arista. C)Vecinos por vértice. D) Vecindad radial empleada por el método.

2.4.1. Algoritmo

##baseMK = $N_{BASE_{MK}}$
##maskNeighbor = $(N_R, N_L = 001)$ $(N_U, N_D = 010)$ $(N_F, N_B = 100)$
##testVal = $(+direction \rightarrow 000)$ $(-direction \rightarrow maskNeighbor)$
##depth = $DL_O(N_{base})$
(resMk,success)← getNeighborMK(baseMK, maskNeighbor,testVal,depth)
{
  ##Init
  test←RSHIFT(LSHIFT(baseMK,2),2);
  success←0;
  resMk←baseMK;
  loop (i←depth..1)
  {

```
    mask←LSHIFT(maskNeighbor,(10-i));
    res←LSHIFT(testVal,(10-i));
    if ((test⋂mask) == res)
    {
      Success←1;
      i←0;
    }
    resMk←(resMk⊕mask);
  }
}
```

El algoritmo anterior solo permite encontrar los vecinos por cara, por lo que para obtener un vecino por arista es necesario aplicar el algoritmo sobre el resultado obtenido, pero en un eje diferente al eje inicial, por consiguiente, para encontrar los vecinos por vértice se tiene que aplicar el algoritmo nuevamente en el eje restante. Empleando este método tenemos como resultado que la búsqueda del subconjunto de nodos que se encuentran dentro de la vecindad radial de una partícula se puede calcular en tiempo $O(c)$ donde la constante c corresponde la cantidad de veces que se aplica el algoritmo para encontrar las MK de los nodos considerando que el tiempo en el que se ejecuta el algoritmo está acotado por una constante que depende del nivel en el que se busca.

El método no requiere el uso de la MK a nivel 10 para realizar el cálculo de la vecindad radial, este cálculo se puede realizar a cualquier nivel de profundidad empleando la $subMK$. Al emplear la $subMK$ se incrementa de manera implícita el radio de búsqueda, pero para mantener la eficiencia del método y obtener un subconjunto de nodos con la menor cantidad de partículas fuera del radio de búsqueda, es necesario considerar que un cambio en un nivel de profundidad, por ejemplo, pasar de nivel 10 a nivel 9 duplica el radio de búsqueda y el radio se continúa duplicando conforme se reduzca el nivel de profundidad.

Demostración

Para demostrar que el método propuesto es correcto hacemos las siguientes consideraciones:

La definición de nodo padre emplea la notación $P(N)$, con lo que se indica que se hace referencia al padre del nodo N, esta operación puede aplicarse de manera reiterativa $P(P(N))$ y solo $P(N_{ROOT}) = \emptyset$.

El algoritmo solo es capaz de devolver la MK del nodo contiguo que se encuentre a la derecha (R), arriba (U) y enfrente (F) con respecto al nodo inicial.

Casos:

1. Vecino derecho. Se verifica con la operación binaria AND entre la clave binaria del nodo inicial y la máscara 001, si el resultado es 000 el nodo se encuentra en la parte izquierda, si es 001 se encuentra en la parte derecha.

    a) Nodo inicial se encuentra en la parte izquierda de su nodo padre.

        1) Siempre existe un vecino y se encuentra contenido en el mismo nodo padre. Dicho vecino se calcula mediante la operación XOR entre la clave binaria del nodo inicial y la máscara 001. Se indica que el vecino se ha encontrado.

    b) Nodo inicial se encuentra en la parte derecha de su nodo padre.

        1) Si existe un vecino se encuentra en la parte izquierda de un nodo padre que sea vecino a la derecha del nodo padre del nodo inicial. Se calcula la clave binaria del nodo mediante la operación XOR entre la clave binaria del nodo inicial y la máscara 001 y se busca el vecino del nodo padre. Si el nivel del nodo inicial es 1 entonces no hay un vecino a la derecha. Si el nivel del nodo inicial es mayor a 1, entonces se repite el proceso de vecino derecha (A) con el nodo padre como nodo inicial.

2. Vecino superior. Se verifica con la operación binaria AND entre la clave binaria del nodo inicial y la máscara 010, si el resultado es 000 el nodo se encuentra en la parte inferior, si es 010 se encuentra en la parte superior.

    a) Nodo inicial se encuentra en la parte inferior de su nodo padre.

        1) Siempre existe un vecino y se encuentra contenido en el mismo nodo padre. Dicho vecino se calcula mediante la operación XOR entre la clave binaria del nodo inicial y la máscara 010. Se indica que el vecino se ha encontrado.

b) Nodo inicial se encuentra en la parte superior de su nodo padre.

1) Si existe un vecino se encuentra en la parte inferior de un nodo padre que sea vecino hacia arriba del nodo padre del nodo inicial. Se calcula la clave binaria del nodo mediante la operación XOR entre la clave binaria del nodo inicial y la máscara 010 y se busca el vecino del nodo padre. Si el nivel del nodo inicial es 1 entonces no hay un vecino hacia arriba. Si el nivel del nodo inicial es mayor a 1, entonces se repite el proceso de vecino arriba (B) con el nodo padre como nodo inicial.

3. Vecino frontal. Se verifica con la operación binaria AND entre la clave binaria del nodo inicial y la máscara 100, si el resultado es 000 el nodo se encuentra en la parte trasera, si es 100 se encuentra en la parte delantera.

a) Nodo inicial se encuentra en la parte trasera de su nodo padre.

1) Siempre existe un vecino y se encuentra contenido en el mismo nodo padre. Dicho vecino se calcula mediante la operación XOR entre la clave binaria del nodo inicial y la máscara 100. Se indica que el vecino se ha encontrado.

b) Nodo inicial se encuentra en la parte delantera de su nodo padre.

1) Si existe un vecino se encuentra en la parte trasera de un nodo padre que sea vecino de enfrente del nodo padre del nodo inicial. Se calcula la clave binaria del nodo mediante la operación XOR entre la clave binaria del nodo inicial y la máscara 100 y se busca el vecino del nodo padre. Si el nivel del nodo inicial es 1 entonces no hay un vecino de enfrente. Si el nivel del nodo inicial es mayor a 1, entonces se repite el proceso de vecino enfrente (C) con el nodo padre como nodo inicial.

Si existe un vecino, dicho vecino tiene la clave Morton del nodo inicial que ha sido modifica en segmentos a través de los niveles en los que se buscó un vecino para dicho nodo.

El algoritmo funciona dado el siguiente lema.

Un nodo N tiene un vecino derecho si el nodo N o alguno de los nodos padres $P(N), P(P(N)), \ldots, P(\ldots P(N))$ en un nivel menor es un nodo localizado en la parte izquierda del espacio comprendido por su nodo padre.

Un nodo N tiene un vecino superior si el nodo N o alguno de los nodos padres $P(N), P(P(N)), \ldots, P(\ldots P(N))$ en un nivel menor es un nodo localizado en la parte inferior del espacio comprendido por su nodo padre.

Un nodo N tiene un vecino frontal si el nodo N o alguno de los nodos padres $P(N), P(P(N)), \ldots, P(\ldots P(N))$ en un nivel menor es un nodo localizado en la parte trasera del espacio comprendido por su nodo padre.

Sea n el número de nodos en Octree con un nivel máximo M, el algoritmo tiene una complejidad lineal O(m) donde m es el nivel del nodo al que se le busca el vecino tal que $m \leq M, m > 0$.

Demostración por contradicción.

Dado un nodo N de nivel M, el algoritmo encuentra un vecino a pesar de que se encuentra en la parte derecha de todos sus nodos padres incluyendo el nodo raíz.

El algoritmo debe terminar e indicar que el vecino se ha encontrado y dado que eso solo es posible si existe un vecino, pero solo se indica que un nodo tiene vecino si el nodo N o alguno de los padres del nodo N se encuentran en la parte izquierda de su padre subsecuente, en caso contrario el algoritmo nunca indica que se encontró un vecino y al llegar al nodo raíz, el programa termina sin encontrar vecino. Por lo tanto, para que encuentre un vecino el nodo debe estar en la parte izquierda dentro de alguno de los nodos padre o debe existir un nivel más allá del nodo raíz donde el nodo padre se encuentre en la parte izquierda.

Esta demostración se extiende por simetría para los casos de búsqueda de vecino arriba y vecino enfrente.

En el apéndice de este trabajo se muestran porciones del código de programación usado para implementar el método descrito en este capítulo, también se incluyen ligas a repositorios donde se encuentra alojado el código para su descarga. La implementación fue realizada en lenguaje C++ y CUDA.

Capítulo 3

Simulación de fluidos

3.1. Método de Hidrodinámica de partículas suavizadas (*Smoothed Particle Hydrodinamics* SPH)

Smoothed Particle Hydrodynamics (SPH) es un método libre de mallas de formulación lagrangiana, el cual fue propuesto y desarrollado por Gingold y Monaghan en 1977 [55]. Este método fue concebido para resolver problemas de astrofísica, sin embargo, dado las similitudes de los sistemas de partículas con el comportamiento de fluidos es posible modelarlo para estos sistemas.

Algunas de las ventajas que presenta SPH sobre los métodos tradicionales de malla presentadas por Liu y Liu [56] son las siguientes:

1. Es un método de naturaleza lagrangiana.
2. Es ideal para el modelado de problemas de superficies libres y flujo con múltiples fases, esto se debe al control que se tiene al desplegar partículas en posiciones especificas en un estado inicial.
3. El método no requiere del uso de mallas.
4. En este método una partícula representa un volumen finito en el espacio continuo.

5. SPH se puede emplear para problemas donde el objeto de estudio no es un continuo.

6. Su implementación numérica es más sencilla que al de los métodos basados en mallas.

En cuanto a la exactitud, estabilidad y eficiencia del método se ha investigado ampliamente, Liu y Liu [57] hace una revisión de esto, un ejemplo de lo anterior es el problema de inconsistencia de partículas reportado por la tesis de Morris [58], lo que resultan en una pobre exactitud, sin embargo, a lo largo de su existencia se han realizado correcciones y cambios con el fin de mejorar dichos aspectos.

3.1.1. Fundamentos de SPH

El método de SPH calcula el valor de diferentes propiedades en cada una de las posiciones donde se encuentran las partículas con las que se representan los objetos de interés, Müller et al. [12] lo define como un método de interpolación para partículas.

Sea A un valor escalar que represente alguna propiedad dentro una región acotada cuya posición es r y r' es la posición en el campo desde donde se mide dicha propiedad, cuyo valor está definido por la ecuación (3.1),

$$A(r) = \int A(r')\delta(r - r')dr' \tag{3.1}$$

donde $\delta(r - r')$ es la función delta de Dirac.

Al sustituir la función δ por el kernel de interpolación $W(r, h)$, donde h es el radio de influencia del kernel, es decir la región de acción efectiva del kernel, obtenemos la integral de interpolación para $A(r)$, esto se muestra en la ecuación (3.2) y es importante mencionar que fuera del radio del kernel la función vale cero.

$$A(r) = \int A(r')W(r - r', h)dr' \tag{3.2}$$

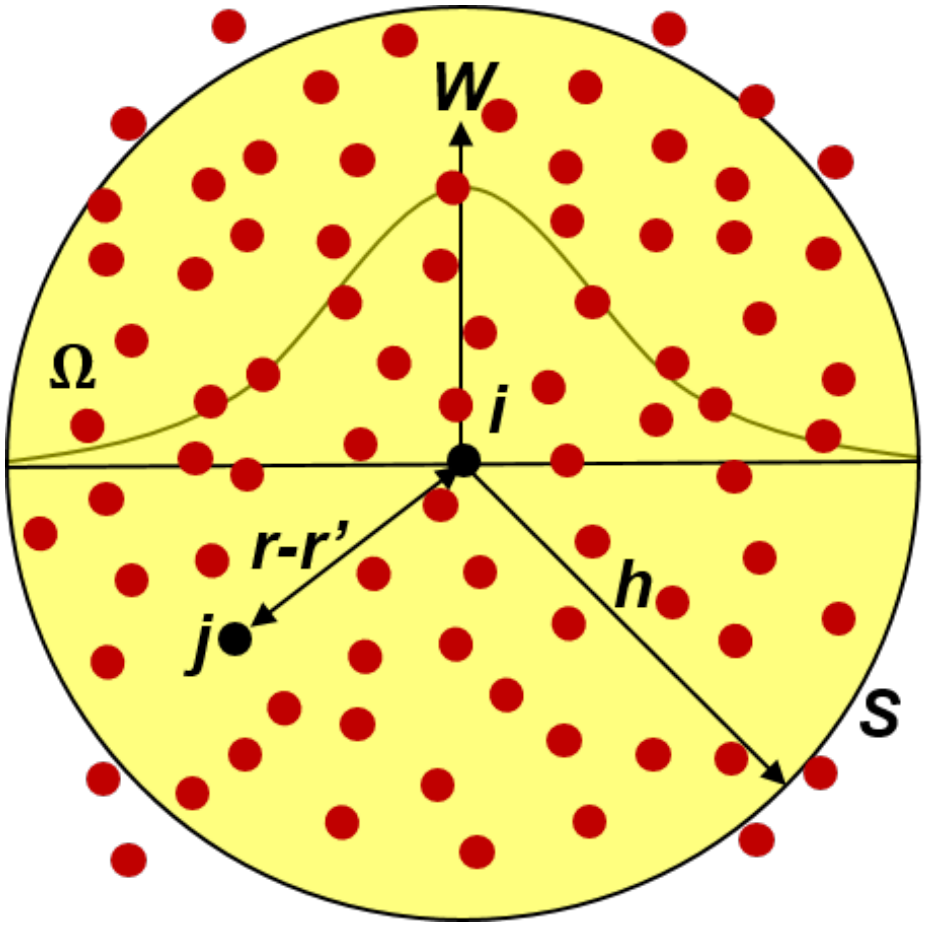

Figura 3.1: Aproximación mediante suma de partículas. El kernel $W(r-r', h)$ es usado para calcular el valor de la propiedad de campo para la partícula i mediante la suma ponderada de cada partícula j que está contenida en la región S.

Las propiedades que el kernel W debe cumplir están definidas por las ecuaciones (3.3) y (3.4).

$$\int W(r - r', h)dr' = 1 \tag{3.3}$$

$$\lim_{h\to 0} W(r - r', h) = \delta(r - r') \tag{3.4}$$

Al modificar la ecuación (3.2) para considerar el sistema de partículas que representa el dominio de interés, se obtiene que el valor de la propiedad de campo A es el resultado de la suma del valor de campo A de cada partícula j, ponderada por el kernel W; lo

anterior se muestra en la ecuación (3.5),

$$A_s(r) = \sum_j A_j \frac{m_j}{\rho_j} W(r - r_j, h), \tag{3.5}$$

donde j itera sobre todas las partículas, m_j es la masa asociada a la partícula j, r_j es la posición en el espacio, ρ_j corresponde a la densidad de la partícula y A_j es el valor de la propiedad de campo calculado en la posición de la partícula.

En la Figura 3.1 se ilustra cómo se calcula el valor de la propiedad de campo A_s, esto se realiza mediante la suma ponderada de las contribuciones de las partículas mediante un kernel de integración $W(r - r_j, h)$.

El método de SPH requiere del cálculo de la densidad ρ_i para todas las partículas i, cada una de estas partículas representa un volumen determinado, además de que tienen una masa constante, sin embargo, la densidad debe ser calculada en cada intercalo de tiempo dando conforme las partículas cambian su posición. Lo anterior se logra mediante la ecuación (3.5), por lo que al sustituir la densidad como propiedad de campo A_j obtenemos como resultado la ecuación (3.6). Hay que indicar que dado lo anterior solo es necesario emplear el valor de la masa para cada partícula m_j.

$$\rho_s(r) = \sum_j \rho_j \frac{m_j}{\rho_j} W(r - r_j, h) = \sum_j m_j W(r - r_j, h), \tag{3.6}$$

3.2. Ecuaciones de Navier-Stokes

Al describir los fluidos isotérmicos mediante la formulación Euleriana, se usan dos ecuaciones para modelar el comportamiento a lo largo del tiempo en conjunto con las propiedades de velocidad v, densidad ρ, viscosidad η y presión $\mathcal{P}$. Las ecuaciones corresponden a conservación de la masa y conservación del momento de Navier-Stokes [59], estas son respectivamente las ecuaciones (3.7) y (3.8).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0, \tag{3.7}$$

$$\rho(\frac{\partial v}{\partial t} + v \cdot \nabla v) = -\nabla \mathcal{P} + \rho g + \eta \nabla^2 v, \tag{3.8}$$

donde g es una fuerza externa debido a la gravedad y ν es el valor de viscosidad asociado al fluido. La ecuación (3.8) es un modelo válido para fluidos Newtonianos incompresibles.

Debido a que una de las consideraciones de SPH es que las partículas no cambian de masa, es posible omitir la ecuación (3.7), esto es gracias a que el método garantiza la conservación de masa mediante las condiciones iniciales del sistema. En la ecuación (3.8), el termino $\frac{\partial v}{\partial t}$ se sustituye por la derivada $\frac{Dv}{Dt}$ y dado que consideramos que las partículas se mueven en conjunto con el fluido no es necesario el cálculo del término $v \cdot \nabla v$.

Si consideramos la parte izquierda de la ecuación (3.8), $\rho\frac{Dv}{Dt}$ como la fuerza f_i, para después emplear la ecuación (3.9), podemos determinar la aceleración a_i de cada partícula i en términos de su velocidad v_i y densidad ρ_i.

$$a_i = \frac{dv_i}{dt} = \frac{f_i}{\rho_i}. \tag{3.9}$$

Al analizar el lado derecho de la ecuación (3.8), es posible identificar tres términos que corresponden a campos de fuerza, una fuerza interna debida a la presión $-\nabla\mathcal{P}$, otra fuerza interna debida a la viscosidad $\eta\nabla^2 v$ y fuerzas externas debido a la gravedad ρg; estos términos los podemos agrupar como se muestra en la ecuación (3.10).

$$f = f^{pressure} + f^{viscosity} + f^{gravity}. \tag{3.10}$$

3.3. Modelo empleado para simulación de fluidos

El modelo empleado para la simulación de fluidos utiliza la definición de propiedad de campo en SPH, está es la que se mencionó en la ecuación (3.5), a partir de dicha ecuación es posible calcular la fuerza debido a la presión y viscosidad; esto se muestra

en las ecuaciones (3.11) y (3.12), las cuales son tomadas de la propuesta de Müller et al. [12]

$$f_i^{pressure} = -\nabla \mathcal{P}(r_i) = -\sum_j \mathcal{P}_i \frac{m_j}{2\rho_j} \nabla W(r_i - r_j, h), \tag{3.11}$$

$$f_i^{viscosity} = \eta \nabla^2 v(r_i) = \eta \sum_j v_j \frac{m_j}{\rho_j} \nabla^2 W(r_i - r_j, h). \tag{3.12}$$

Las ecuaciones (3.11) y (3.12) tienen la característica de que la fuerza resultante no es simétrica, esto genere problemas de estabilidad, por lo que se requiere hacer un ajuste a dichas ecuaciones. La ecuación resultante es (3.13), la cual determina la fuerza debido a la presión mediante la suma de la media aritmética de las presiones generadas entre cada par de partículas que se encuentran interactuando entre sí; mientras que la ecuación (3.14) determina la fuerza debida a la viscosidad como una fuerza que actúa en dirección al cambio de velocidad relativo de su marco de referencia, el cual es definido por su vecindad.

$$f_i^{pressure} = -\sum_j (\mathcal{P}_i + \mathcal{P}_j) \frac{m_j}{2\rho_j} \nabla W(r_i - r_j, h), \tag{3.13}$$

$$f_i^{viscosity} = \eta \sum_j (v_j - v_i) \frac{m_j}{\rho_j} \nabla^2 W(r_i - r_j, h). \tag{3.14}$$

Para calcular el valor debido a la presión $\mathcal{P}$ se emplea una variante de la fórmula propuesta por Desbrun y Gascuel [28], su propuesta emplea el valor de la diferencia de la densidad base ρ_0 y la densidad de la partícula mediante su valor de campo $\rho = \rho_s(r)$, esta diferencia se pondera por el factor de presión k_p, el cual depende de la temperatura de las partículas; esto se muestra en la ecuación (3.15).

$$\mathcal{P} = k_p(\rho - \rho_0). \tag{3.15}$$

Modelando solo viscosidad y densidad no es posible simular diferentes tipos de fluidos, por ejemplo, agua y sangre, miel y jarabe; por lo que es necesario incluir en

el modelo la tensión superficial. Al agregar la tensión superficial es posible modelar diferentes comportamientos, aunque los fluidos tengan presión y viscosidad similares.

La tensión superficial es una fuerza originada por la cohesión entre las moléculas que conforman un fluido, esta fuerza se comporta de manera elástica deformando la superficie del fluido a fin de tener la menor área superficial posible.

Un modelo ampliamente aceptado para el cálculo de la tensión superficial es el de Müller et al. [14], este modelo emplea un valor conocido como campo de color c_s, en la ecuación (3.16) se muestra la fórmula empleada para calcularlo.

$$c_s(r_i) = \sum_j m_j \frac{1}{\rho_j} W(r_i - r_j, h). \tag{3.16}$$

Las fuerzas debidas a la tensión superficial son normales a la superficie, por lo que se emplea el gradiente del campo de color para obtener la normal a la superficie del fluido, $\overline{n} = \nabla c_s$, además de considerar la curvatura de la superficie, $\kappa = -\frac{\nabla^2 c_s}{|\overline{n}|}$. Se debe tomar en cuenta que está fuerza solo se presenta en las partículas cercanas a la superficie, por lo que es necesario determinar está distancia α. La ecuación (3.17) muestra la formula para calcular la fuerza debida a la tensión superficial $f^{surface}$.

$$f^{surface} = \alpha \kappa \overline{n} = -\alpha \nabla^2 c_s \frac{\overline{n}}{|\overline{n}|}. \tag{3.17}$$

Existen propuestas para el cálculo de esta fuerza, como la presentada por Li et al. [60] y Shao et al. [61], que ofrecen un comportamiento más cercano a la realidad, sin embargo son un poco más costosas de calcular, por lo que para una simulación cuyo objetivo principal es conseguir despliegue en tiempo real es un preferible sacrificar un poco de realismo siempre que se mantenga el desempeño adecuado. Por otro lado, los métodos anteriormente mencionados parten de conceptos similares, como el campo de color y curvatura de la superficie, aunque estos métodos agregan interacción entre diferentes fases, lo hacen de manera binaria.

3.4. Kernel de suavizado (Smoothing Kernel)

El método de SPH depende en gran medida de los kernel de suavizado que se usen, estos tienen incidencia directa en la estabilidad, exactitud y velocidad de los cálculos que se realizan. Las ecuaciones (3.18), (3.19) y (3.20), definen 3 tipos de kernel de suavizado para SPH, estos kernel han sido ampliamente usados en lo reportado por Müller [12], [14], Macklin [13], [15] y Chen [17]. En la literatura existen diferentes tipos de kernel sin embargo comparte algunas similitudes, Liu [56] hace un análisis de cómo construir dichos kernel, mientras que Takahashi [16] propone el uso de un kernel híbrido para mejorar la robustez de las colisiones de fluidos y fluidos con sólidos.

$$W_{poly6}(r,h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3, & 0 \leq r \leq h \\ 0, & r > h \end{cases}, \tag{3.18}$$

$$W_{spiky}(r,h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3, & 0 \leq r \leq h \\ 0, & r > h \end{cases}, \tag{3.19}$$

$$W_{visc}(r,h) = \frac{15}{2\pi h^3} \begin{cases} (-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1, & 0 \leq r \leq h \\ 0, & r > h \end{cases}. \tag{3.20}$$

En la Figura 3.2 se muestran las gráficas de los kernels descritos anteriormente, en cada una de las secciones se tiene la función del kernel W, su gradiente ∇W y su laplaciano $\nabla^2 W$, las funciones mostradas no están normalizadas y su propósito es únicamente mostrar su tendencia con respecto a la distancia; se estableció el valor $h = 1{,}0$ y se escalaron los valores del eje Y para que se pueda observar todo en una sola gráfica.

Müller propone las propiedades, $W(|r| = h, h) = 0$ y $\nabla W(r = h, h) = 0$; las anteriores tienen el fin de mejorar la estabilidad del método, al mismo tiempo evitan agregar amortiguamiento a la fuerza debido a la viscosidad. De manera adicional a dichas propiedades, se emplea la ecuación (3.21) para calcular el laplaciano, este es empleado para calcular el valor de campo de viscosidad de la ecuación (3.14).

$$\nabla^2 W(r,h) = \frac{45}{\pi h^6}(h - r). \tag{3.21}$$

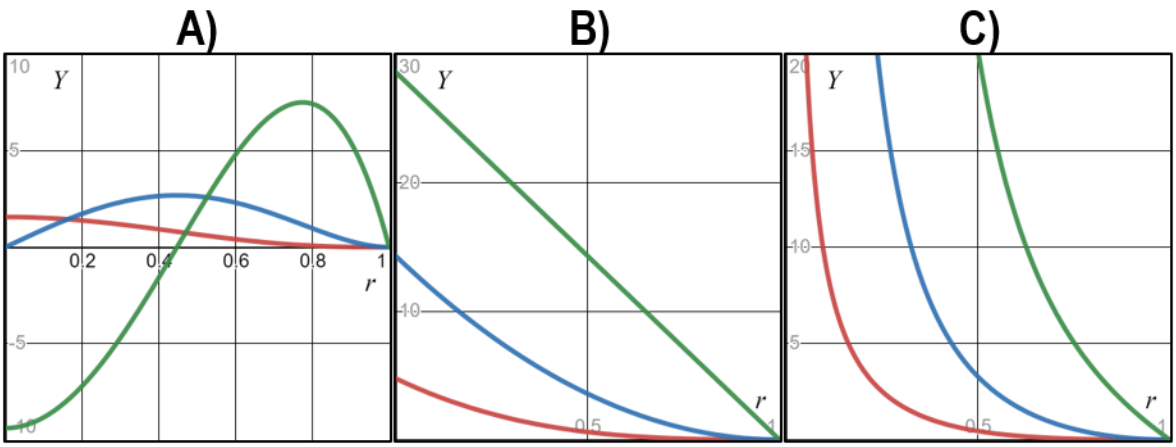

Figura 3.2: Gráficas de kernels W, en rojo W, en azul ∇W, en rojo $\nabla^2 W$. A) Kernel poly6. B) Kernel spiky. C) Kernel visc.

3.5. Implementación de Simulación de fluidos en conjunto con Octree

En este trabajo se propone hacer uso de las ecuaciones definidas en la sección anterior y el método de búsqueda de vecinos propuesto con anterioridad. Con base a esto es necesario tener claro que la simulación debe definir ciertos valores en función de las dimensiones del espacio de trabajo, el nivel de profundidad del Octree y la cantidad de partículas que pueden ser contenidas en los nodos de máximo nivel del Octree.

En el método propuesto se hacen las siguientes consideraciones iniciales:

- El espacio de trabajo de trabajo tiene 3 dimensiones (L_x, L_y, L_z) donde $L_x = L_y = L_z$. Esto asegura que los nodos del Octree sean cubos perfectos, lo anterior no es necesario para realizar la división del espacio con indexado mediante claves Morton, sin embargo, de este modo se asegura consistencia en radios de búsqueda y control en el volumen de partículas pare contenidas dentro de un nodo.
- Nivel de profundidad máximo de un nodo es 10. Aunque el método de división del espacio no especifica un límite, por factores de memoria y construcción de clave Morton se ha decidido acotarlo a nivel 10, es posible incrementar el nivel, pero para fines de este trabajo se ha acotado hasta dicho nivel.

Con las consideraciones anteriormente mencionadas podemos calcular el radio de las partículas p_{rd} con base a la cantidad de partículas n_{pt} que un nodo, que se encuentra en el nivel de profundidad máximo N_M^{id}, puede contener. Se debe tener en cuenta que el volumen del nodo, cubo, se debe aproximar a la suma del volumen de las partículas, esferas; por lo que dicho cubo tiene las dimensiones $l_x = \frac{Lx}{2^M}$, $l_y = \frac{Ly}{2^M}$ y $l_z = \frac{Lx}{2^M}$. Usando la ecuación (3.22) se calcula el radio de las partículas cumpliendo las condiciones anteriormente enunciadas.

$$p_{rd} = \frac{l_x}{\sqrt[3]{\frac{n_{pt}4\pi}{3}}}. \tag{3.22}$$

Al determinar el valor de p_{rd} con la ecuación anterior se tiene la certeza de que no es posible tener más partículas dentro de un nodo del límite fijado, en la Figura 3.3 se presenta mediante un diagrama el caso ideal, donde 8 partículas ocupan todo el volumen de un nodo. Sin embargo, a fin de que esta condición sea cumplida es recomendable que también se cumpla que el valor sea un número entero, es decir $\sqrt[3]{n_{pt}} \in \mathbb{Z}$.

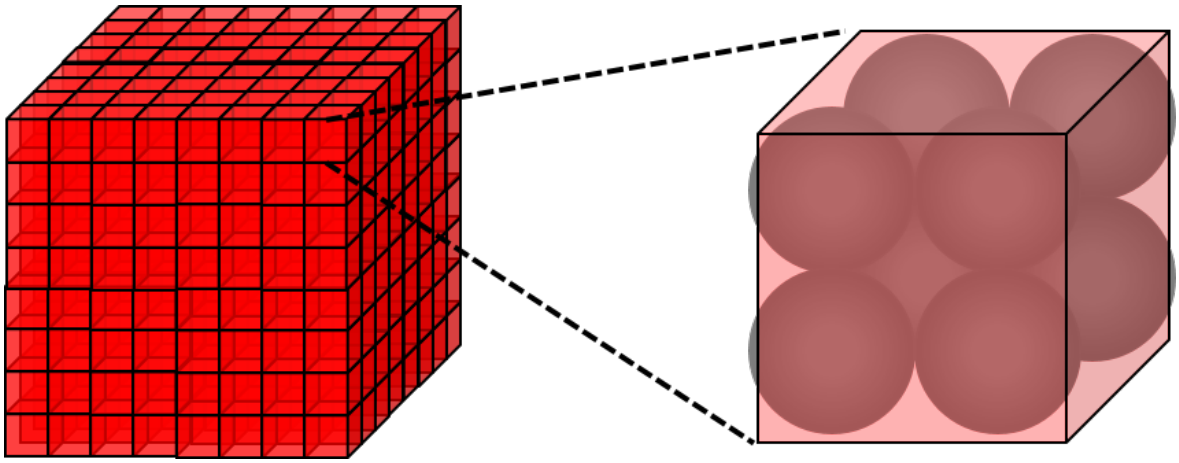

Figura 3.3: Volumen de un nodo siendo ocupado en su totalidad por 8 partículas.

El valor del radio de acción del kernel h determina en que región las partículas son consideradas para los cálculos de SPH, sin embargo, dependiendo de las dimensiones de los nodos (l_x, l_y, l_z) a nivel máximo M, es posible que el valor h sea mayor que la región donde el método busca vecinos. Debido a esta característica es que es necesario calcular un valor de límite de vecindad h', para el cual es recomendable que cumpla con la propiedad $h' < h$. La ecuación (3.23) se emplea para determinar el valor del límite de vecindad h', este valor se determina usando las dimensiones de los nodos de

nivel máximo de profundidad del Octree.

$$h' = l_x = \frac{L_x}{2^M}. \tag{3.23}$$

Por último, se determinó la velocidad y aceleración máxima de cada partícula, esto está en función del radio de interacción y los cuadros por segundo que se desean obtener, para nuestro fin asumimos una velocidad de despliegue de 60 a 240 cuadros por segundo. La ecuación (3.24) y (3.25), muestra el cálculo de la velocidad y aceleración máximas de las partículas.

$$u_{MAX} = \frac{h'}{\Delta t}. \tag{3.24}$$

$$a_{MAX} = \frac{h'}{\Delta t^2}. \tag{3.25}$$

Capítulo 4

Simulación de cuerpos suaves y colisiones en sistemas de partículas

4.1. Cuerpos deformables con SPH

A diferencia de los fluidos, los cuales al ser representados mediante un sistema de partículas donde cada partícula actúa sobre las demás partículas que la rodean y estás se pueden mover de manera libre a lo largo de todo el dominio, los cuerpos sólidos, en específico los elementos de los cuerpos deformables, no cumplen esa condición. Dado lo anterior es que el enfoque predominante en las simulaciones de este tipo son dominadas principalmente por enfoques basados en métodos de elemento finito, FEM por sus siglas en inglés Finite Element Method.

En la bibliografía existen trabajos que demuestran que las ecuaciones que modelan el comportamiento elástico de los cuerpos solidos pueden ser resueltas mediante el método de SPH, por ejemplo el trabajo de Cleary [62], sin embargo, para evitar problemas de inestabilidad a la tracción es necesario agregar fuerzas externas entre pares de partículas, esto es mencionado en lo reportado por Monaghan [63] Y Gray [64]. Al tener en cuenta dicha condición, es que para este trabajo se optó por crear restricciones entre pares de partículas para así facilitar el modelado del comportamiento viscoelástico.

4.2. Modelo viscoelástico

Uno de los modelos mas complejos a simular es el de los cuerpos viscoelásticos, existen varias aplicaciones que requieren de este tipo de modeloado, por ejemplo, los tejidos del cuerpo humano tienen propiedades elásticas, viscosas y plásticas, en especial las estructuras vasculares, como son venas y arterías, son viscoelásticas.

La simulación de cuerpos deformables, como lo son los tejidos suaves en simuladores quirúrgicos o representación de diversos objetos deformables en videojuegos, tiene un rol importante en la recreación de una experiencia realista.

El modelado de cuerpos deformables considera que los objetos son sólidos con propiedades viscoelásticas y plásticas. Podemos separar cada una de estas propiedades y analizarlas de manera individual para posteriormente conjuntarlas para obtener un modelo para materiales suaves. En este trabajo únicamente nos enfocamos en el modelado de sólidos mediante sus propiedades viscoelásticas.

La elasticidad lineal se modela mediante la ley de Hooke, donde la fuerza F es proporcional a la deformación lineal de un cuerpo X y el coeficiente de rigidez del material del que está echo el cuerpo μ_0. En la ecuación (4.1) corresponde a la ley de Hooke.

$$F = \mu_0 X. \tag{4.1}$$

Un modelo simple de la viscosidad lineal se obtiene al considerar el objeto como un flujo viscoso, por lo que podemos considerar que la velocidad de deformación $\bar{X}$ multiplicada por su coeficiente de viscosidad η_0 es la fuerza aplicada al cuerpo F. La ecuación (4.2) corresponde a lo descrito anteriormente.

$$F = \eta_0 \bar{X}. \tag{4.2}$$

Reescribiendo las ecuaciones (4.1) y (4.2) en términos del estrés σ y tensión ε tenemos las ecuaciones (4.3) y (4.4), donde el coeficiente de rigidez es remplazado por el módulo de elasticidad, también conocido como módulo de Young E . Estas ecuaciones representan el modelo de un resorte y un amortiguador respectivamente.

$$\sigma(t) = E\varepsilon(t), \tag{4.3}$$

$$\sigma(t) = \eta_0\dot{\varepsilon}(t) = \eta_0\frac{\varepsilon(t)}{dt}. \tag{4.4}$$

Los modelos de viscoelasticidad empleados para simular cuerpos deformables son el modelo de Maxwell, Kelvin-Voigt y Solido lineal estándar, SLS por sus siglas en inglés Standard Lineal Solid, en la Figura 4.1 se muestra un diagrama de los modelos mencionados.

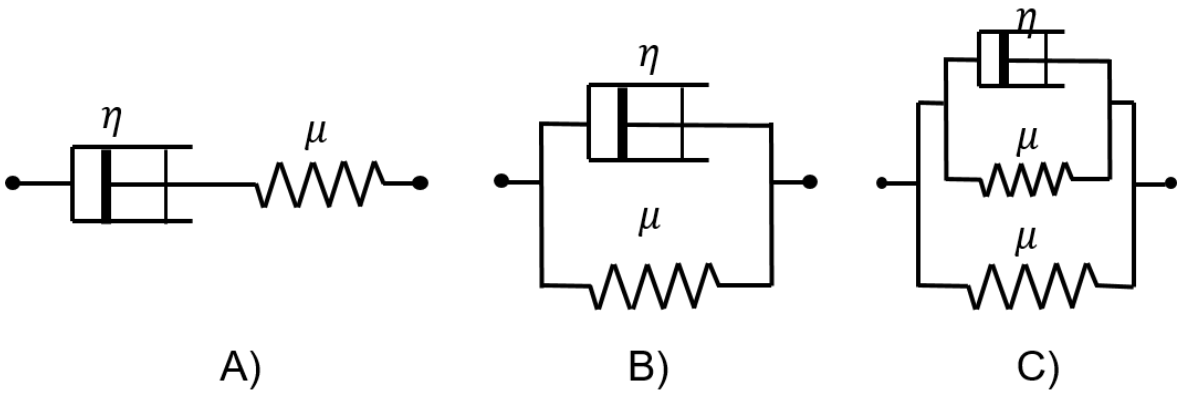

Figura 4.1: Modelos de deformación viscoelásticos. A) Maxwell. B) Kelvin-Voigt. C) SLS.

Cada uno de estos modelos tiene diferentes propiedades. El modelo de Maxwell considera que el estrés en cada elemento es el mismo y este es igual al estrés total del sistema, mientras que la tensión total es igual a la suma de las tensiones en cada elemento. El modelo de Kelvin-Voigt considera que el estrés total del sistema es igual a la suma del estrés en cada elemento y la tensión total es igual a la tensión de cada elemento. Las ecuaciones (4.5), (4.6) y (4.7) corresponden a las ecuaciones constitutivas de modelos de Maxwell, Kelvin-Voigt y SLS.

$$\frac{d\varepsilon(t)}{dt} = \frac{\sigma}{\eta_0} + \frac{1}{E}\frac{d\sigma}{dt}, \tag{4.5}$$

$$\sigma(t) = E\varepsilon(t) + \eta_0\frac{d\varepsilon(t)}{dt}, \tag{4.6}$$

$$(E_1 + E_2)\frac{d\varepsilon}{dt} = \frac{E_2}{\eta_0}\left(\frac{\eta_0}{E_2}\frac{d\sigma}{dt} + \sigma - E_1\varepsilon\right). \tag{4.7}$$

Al emplear el modelo anterior sobre un volumen se requiere definir un tensor de estrés σ_{ij}, el tensor de estrés se define como la fuerza por área en cierta dirección sobre la superficie del cuerpo. La ecuación (4.8) define el tensor de estrés de 9 componentes.

$$\sigma_{ij} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}. \tag{4.8}$$

Las componentes de la diagonal principal son las componentes de estrés normales, mientras que las 6 restantes son las componentes de corte (shear). En el caso de las que no exista un torque sobre el cuerpo, la matriz es simétrica sobre la diagonal. Se puede realizar una simplificación mediante la condición de que las componentes de corte son nulas.

4.3. Colisión entre partículas

Los métodos más populares para la detección de colisiones encontrados en la literatura emplean cajas englobantes, es decir que emplean subdivisión espacial para acelerar el cálculo de vecindades y así detectar mediante distancia el evento de colisión. En el trabajo de Tu y Yu [65], se realiza un análisis de los métodos que emplean cajas englobantes como son cajas englobantes alineadas a los ejes (Axis-Aligned Bounding Box - AABB), caja esfera (Sphere Box), caja englobante de orientación (Orientation Bounding Box – OBB) y K-Discretos politopos de orientación (K-Discrete Orientation Polytopes); en este mismo trabajo se presenta un método hibrido entre AABB y OBB para acelerar el cálculo de colisiones.

Aunque los métodos como AABB y OBB son populares tienen como limitación que están diseñados para detectar colisiones entre un número limitado de objetos, esto es mencionado por Sulaiman [66] y Yong [67]. Esta característica limita su utilidad para sistemas de partículas, donde todos los elementos están formados por una gran cantidad de partículas.

Las propuestas que existen para modelar colisiones entre partículas y objetos con cajas de colisión emplean un proceso de voxelización, este proceso, aunque trivial agrega una caja de colisión a las partículas, lo que regresa el problema a algo similar a AABB y OBB, esto se muestra en el trabajo de He [18], Tang [68] y Yong [67]. Aunque los métodos presentados por los mencionados anteriormente ofrecen ventajas sobre AABB y OBB, no tienen un enfoque libre de mallas. Estos métodos colisionan superficies contra partículas, lo cual no es nuestro objetivo.

En el trabajo de Müller [14] y Kolb [69], se presentan métodos que las colisiones son calculadas directamente entre partículas, sin embargo, existen algunos detalles que no permiten emplearlos tal como fueron formulados para sistemas completamente basados en partículas. En el trabajo de Müller se sustituyen mallas triangulares por un conjunto de partículas, a las cuales se asocian las propiedades de dicha superficie y la colisión emplea el cálculo de un valor similar al potencial de Lennard-Jones $V(r)$, en la ecuación (4.9) se muestra la fórmula para calcular dicho potencial, que es la fuerza que actúa entre un par de partículas, como son átomos o moléculas.

$$V(r) = 4\varepsilon_0 \left[(\frac{\sigma_0}{r})^{12} - (\frac{\sigma_0}{r})^6 \right]. \tag{4.9}$$

Donde ε_0 es la profundidad del potencial, σ_0 es la distancia mínima en la que el potencial es válido y r es la distancia entre las partículas.

En el trabajo de Klob, también se recurre al cálculo de una superficie implícita de los límites de los objetos, con la cual se calcula la normal a la velocidad en los eventos de colisión. Sin embargo, el enfoque por velocidad es útil para nuestro fin, ya que permite distinguir entre objetos que están por colisionar y los que ya han colisionado.

4.3.1. Método de detección de colisiones propuesto

El método que proponemos en este trabajo no requiere de un cálculo previo de una superficie, esta es una condición propuesta ya que la superficie es representada de manera indirecta por un conjunto de partículas. En la figura 4.2, se muestra un diagrama donde un vector $\vec{v_1}$ es reflejado al hacer contacto sobre una superficie, la cual tiene una normal $\hat{n}$ sobre el punto de incidencia, lo cual da como resultado un vector

$\vec{v_2}$; los ángulos Θ_1 y Θ_2 son iguales.

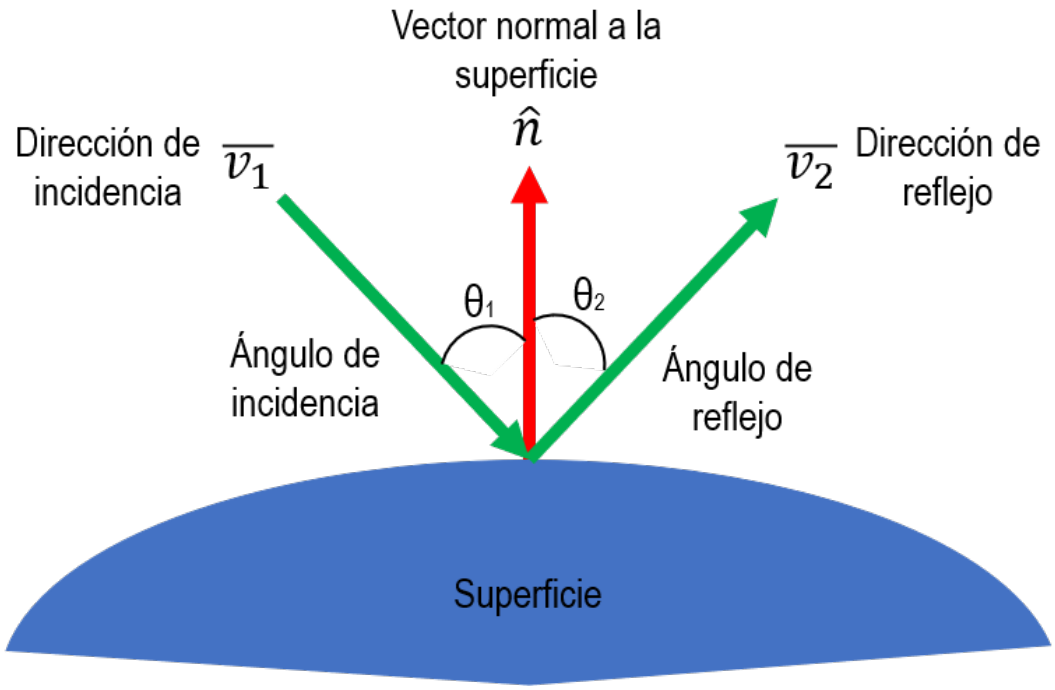

Figura 4.2: Diagrama de resultado de colisión sobre una superficie donde se considera la normal a la superficie $\hat{n}$, dirección y ángulo de incidencia, $\vec{v_1}$ y Θ_1; esto da como resultado una dirección y ángulo de reflejo, $\vec{v_2}$ y Θ_2.

Aunque el cálculo de la superficie implícita ha sido ampliamente estudiado en otros trabajos, hemos decidido dejarlo fuera de las labores del motor de simulación desarrollado. Por lo que la colisión que usaremos se evalúa únicamente entre partículas. Este proceso es eficiente dado que empleamos el mismo proceso de cálculo de vecindades que ha sido optimizado usando el Octree indexado por claves Morton.

El método evalúa primero si las partículas forman parte de la vecindad radial donde existe una colisión. Para evitar problemas de interpenetración y no detección de colisiones, se debe tomar en cuenta el tamaño de cada partícula y hacer un ajuste a la velocidad del sonido respecto a este radio de colisión.

El método determina si dos partículas se encuentran en proceso de colisión empleado el criterio de reducción distancia entre ambas partículas. Una manera simple para determinar esto es calcular el producto punto entre la velocidad de una partícula y el vector de dirección desde la segunda partícula a la partícula inicial. Sí este valor es negativo las partículas se encuentran en proceso de colisión, lo dicho anteriormente se

muestra en la ecuación (4.10).

$$\hat{r}_{ji} \cdot \vec{v}_i < 0, \tag{4.10}$$

Un problema con esta propuesta es que todo acercamiento entre partículas puede resultar en una colisión, lo cual no es ideal, así que para eso tenemos que medir el ángulo de colisión, para obtener este ángulo empleamos la diferencia en la dirección de movimiento y la dirección entre las partículas. En la ecuación (4.11) se muestra lo dicho anteriormente.

$$\hat{r}_{ji} \cdot \frac{\vec{v}_i}{|\vec{v}_i|} > \epsilon, \tag{4.11}$$

Si se cumplen las condiciones mencionadas anteriormente la colisión se produce y se resuelve mediante el cálculo de una nueva dirección de movimiento de la partícula incidente. Esta dirección es la acumulación de inversa dirección de incidencia sobre el conjunto de partículas contra las que se colisiona, ponderada por la magnitud de la velocidad, esta suma se promedia y pondera para simular amortiguamiento. En la ecuación (4.12) se muestra lo dicho anteriormente.

$$\vec{v}_{i,col} = \frac{k_c}{n} \sum_{j}^{n} |\vec{v}_i| \hat{r}_{ji}. \tag{4.12}$$

El método propuesto asemeja una propiedad de campo de SPH, sin embargo, no emplea las propiedades de campo. En la figura 4.3 se ilustran los escenarios donde se produce una colisión y donde la colisión no es considerada. Para ellos se evalúa el resultado del producto escalar entre la velocidad $\vec{v}_i$ de una partícula p_i y la dirección $\hat{r}_{ji}$ desde la partícula p_j que sobre la que incide.

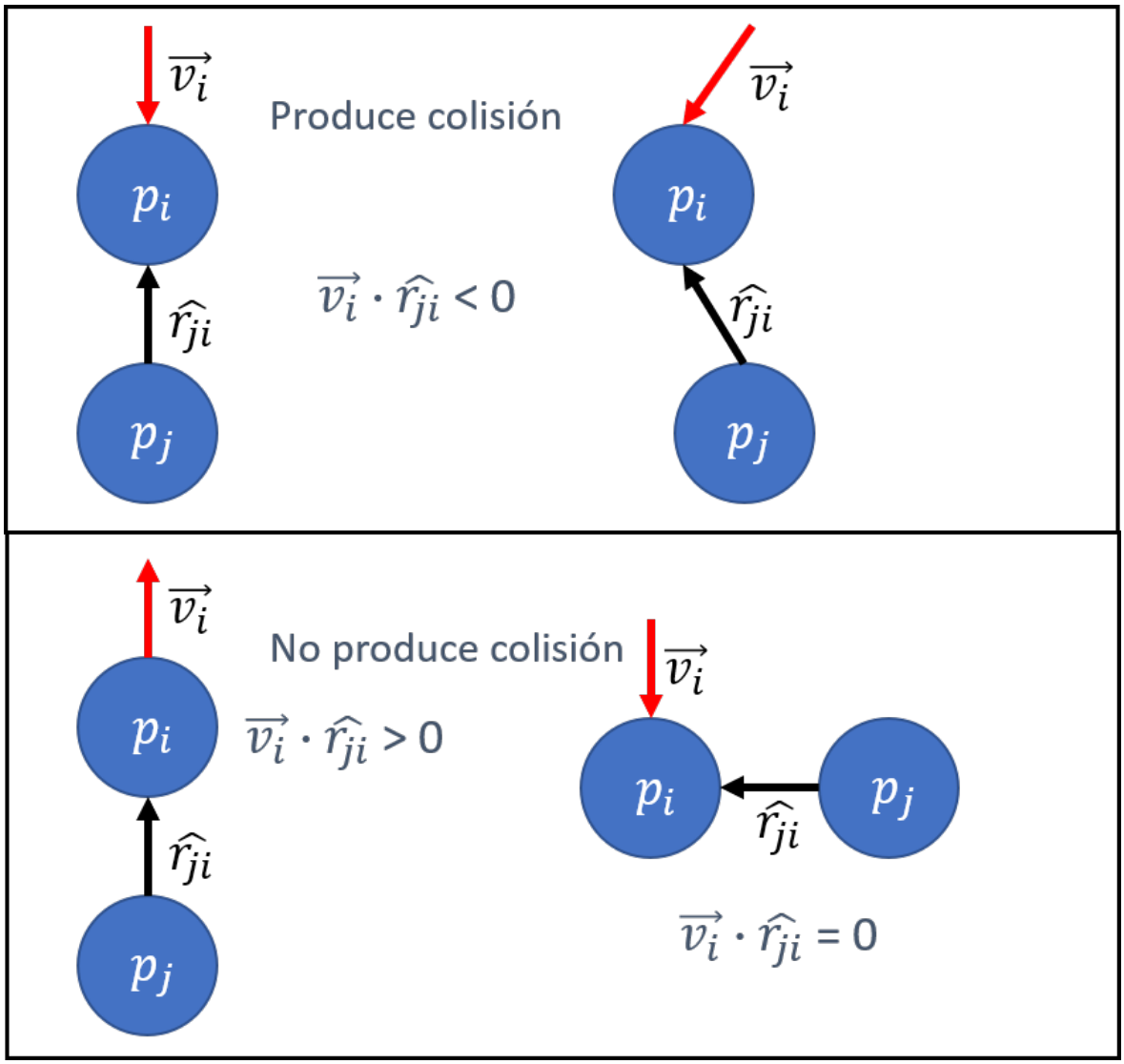

Figura 4.3: Escenario de detección y no detección de colisiones entre partículas. Hay colisión cuando el producto escalar es menor a 0, mientras que con un valor igual o mayor que 0 no se considera la existencia de colisión.

Capítulo 5

El problema de *Big Data* en el manejo de información en sistemas de partículas libres

En la actualidad se ha presentado un fenómeno de crecimiento de datos de manera exponencial, por lo que se ha popularizado el uso del término de *Big Data* para denominar a estos grandes cúmulos de información que son producidos de manera cotidiana. La tecnología actual permite generar y almacenar estos grandes volúmenes de información, sin embargo, existen desafíos para su utilización.

Debido a que el volumen de datos en *Big Data* es enorme, se han desarrollado técnicas enfocadas en la minería de datos, en inglés se conoce como *Data Mining*, mediante los cuales se proponen modelos para el procesamiento de *Big Data*. Un ejemplo de esto es el trabajo de Wu et al. [70], donde se presenta el teorema HACE para caracterizar los rasgos de la revolución de *Big Data*.

Aunque la minería de datos permite el estudio de la información contenida en *Big Data*, este proceso parte del supuesto de que es posible almacenar en memoria y procesar al mismo tempo toda la información disponible, o en su defecto, toda la información relacionada al caso de estudio que se esté analizando. Esto presenta un desafío para enfoques que emplean herramientas como aprendizaje de máquina, en inglés se conoce como *Machine Learnign*, lo anterior es abordado en el trabajo de

L'Heureux et al. [71], donde se hace notar, que al usar tecnologías como las GPU para acelerar el proceso de *Machine Learning*, no se dispone de los mismos recursos de memoria, que sí están disponibles en otras tecnologías.

En la revisión presentada por Soriano et al. [44], se conjuntan las áreas de *Big Data*, *Machine Learning* y análisis de datos enfocados a la práctica clínica. Con base en el trabajo mencionado, es que encontramos vínculos entre diferentes áreas, uno de ellos es la simulación y *Big Data*. En el caso de los métodos de simulación libres de mallas, como los presentados en este trabajo, existe una relación importante entre el número de partículas, resolución del método y tamaño del dominio (volumen de trabajo).

5.1. *Big Data* y SPH

Es habitual tener modelos que se representan mediante sistemas con una cardinalidad mayor a las centenas de miles de partículas, donde además los intervalos de tiempo para su estudio se definen en millonésimas de segundo; por lo que al ser posible capturar y almacenar toda esta información manteniendo la representación lagrangiana, en la cual se proporcionan los datos producidos por estos sistemas en cada intervalo de tiempo, tenemos como resultado un volumen de información que supera fácilmente los millones o miles de millones de datos por segundo.

Un enfoque de las simulaciones que empelan partículas libres es el despliegue gráfico inmediato a una pantalla del resultado obtenido de la simulación. Un ejemplo de esto son los videojuegos, donde el manejo de la información correspondiente a las partículas se realiza mediante un motor de renderizado gráfico, como lo son OpenGL, Vulcan o Direct x, por mencionar algunos. Para estos casos, el uso del método propuesto en este trabajo permite acelerar los cálculos, tal que es posible ejecutar simulaciones de sistemas de partículas a una taza de refresco con el cual se mantiene la inmersión visual, 60 cuadros por segundo.

Sin embargo, existen aplicaciones con otro enfoque, en el cual el objetivo es obtener descriptores de modelos con los que es posible medir el comportamiento de diferentes fenómenos en momentos específicos, por ejemplo, el trabajo de Dal Ferro et al. [41].

En casos como el mencionado anteriormente, se requiere un método para presentar

de una manera entendible y medible los resultados de la simulación. Existen diversos métodos para lograr esto, por ejemplo, la representación de cada valor de cada una de las partículas en cada instante di tiempo como un campo escalar o la representación del sistema de partículas en una forma euleriana, donde en lugar de de tener un valor por cada partícula, se tiene un valor por puntos arbitrarios en el espacio, los cuáles se encuentran en el volumen de interés.

Un problema de este enfoque es el manejo eficiente del volumen de interés, pero con el método propuesto en este trabajo, es posible realizar las operaciones necesarias de manera eficiente, mediante el uso de una arquitectura en paralelo, para obtener los valores medidos en cada una de las regiones de interés, usando los nodos del Octree. Además, el uso del Octree propuesto, permite presentar los resultados de la simulación a diferentes resoluciones, empleando únicamente los niveles de profundidad del Octree.

El poder manejar diferentes resoluciones, es una característica indispensable, ya que la simulación de diferentes fenómenos requiere reportar datos a diferentes resoluciones, la cual, no es necesariamente igual a la empleada para representar los modelos dentro de la simulación.

En la Figura 5.1 se muestra un ejemplo de salida gráfica de los datos producidos por una simulación de SPH utilizando los nodos del Octree a niveles 9, 8 y 7, se gráfica el valor de velocidad máxima en cada uno de los nodos. En este ejemplo se simularon 926 cuadros, resultando en 49,197,842 mediciones de partículas a lo largo de un periodo de tiempo arbitrario.

Por otro lado, si los datos los desplegamos empleando únicamente una metodología lagrangiana, solo podemos obtener información en las regiones donde tenemos partículas y solo durante el periodo en el que las partículas se encuentran en dicha posición. En la Figura 5.2 se muestra la secuencia de simulación de flujo que se mostró en la figura anterior, pero con la diferencia de que se despliega la información de modo totalmente lagrangiana.

El método nos permite diferentes formas de trabajo, por ejemplo es posible presentar las propiedades de los campos escalares mediante la resolución de los nodos del Octree. En la Figura 5.3, se presenta una porción de una nube de puntos con 964,933 elementos, la cual representa la fotogrametría de un hueso con una marca sobre su superficie. La

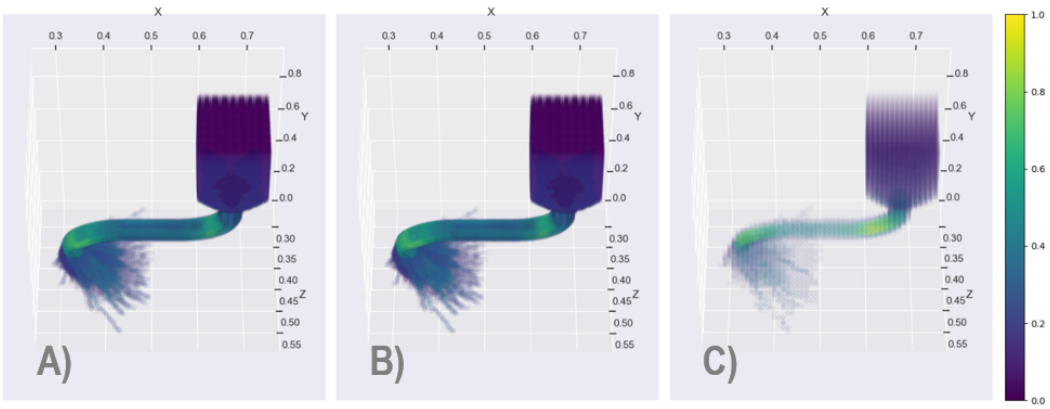

Figura 5.1: Ejemplo de simulación de flujo en tubo, presentada con Octree a diferente nivel de profundidad. A) Nivel 9 con despliegue de 141,114 elementos. B) Nivel 8 con despliegue de 117,209 elementos. C) Nivel 7 con despliegue de 11,503 elementos.

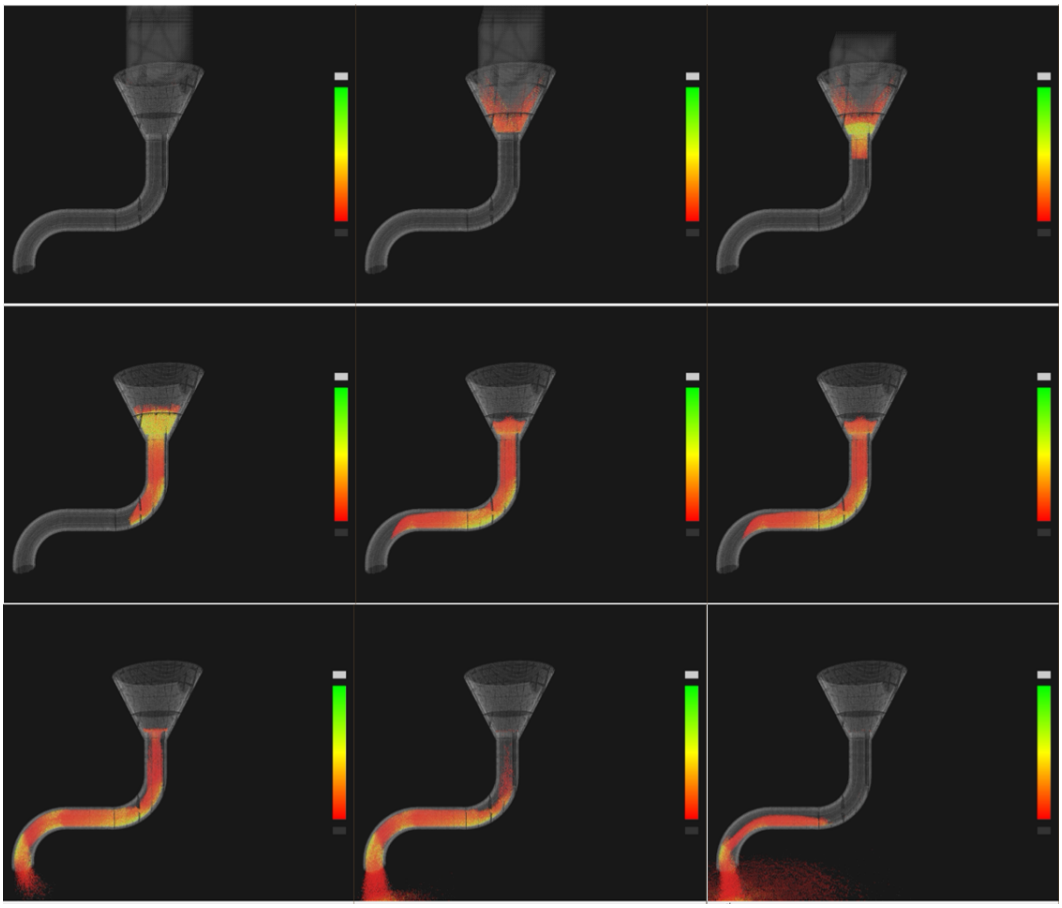

Figura 5.2: Ejemplo de simulación de flujo en tubo empleando visualización de datos de presión en fluido con metodología lagrangiana.

nube de puntos está embebida en un Octree con nivel de profundidad máxima 7. Con esta configuración es posible medir propiedades de la superficie formada por una nube de partículas. Empleando el método de cálculo de vecindades propuesto en este trabajo, es posible obtener un valor asociado a la curvatura direccional para cada partícula y al modificar la representación mediante el nivel de los nodos, se puede obtener información a múltiples resoluciones, teniendo así información de detalles o de forma en general.

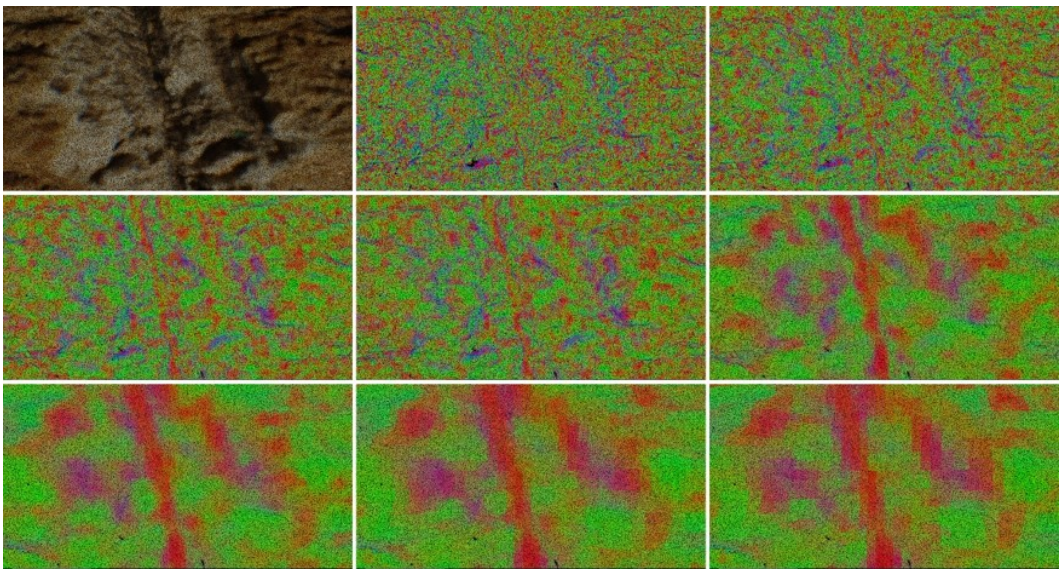

Figura 5.3: Aplicación de medida de forma, empleando el valor de cambio de normal asociada a la superficie, usando kernels de SPH con diferentes niveles de resolución de Octree.

Capítulo 6

Experimentos y Resultados

A continuación, se muestran los experimentos propuestos para evaluar el desempeño del método propuesto en este trabajo. En vista de que el objetivo es analizar el desempeño de este método, se hace análisis de su rendimiento y comportamiento al ser ejecutado en arquitecturas en paralelo mediante Unidades de procesamiento gráfico de propósito general o GPGPU por las siglas en ingles de General-Purpose Graphics Proccesing Unit.

6.1. Exp 1. Calculo de vecindad

El experimento 1 ha sido planteado para evaluar el desempeño del método de búsqueda de vecinos contenidos en los nodos adyacentes a una partícula. En el cálculo de la vecindad se empleó el método propuesto de el Octree indexado mediante claves Morton.

Las especificaciones del equipo de cómputo empleado para realizar este experimento son las siguientes:

- Procesador: Intel i7-7700HQ.
- GPU: Nvidia GeForce GTX 1070 8GB.
- Memoria: 16GB RAM.

Para el cálculo de vecindad radial se empleó el mismo algoritmo en todos los casos, sin embargo, se realizaron variaciones en cuanto a la implementación de la estructura de datos, esto tiene como objetivo observar cual aprovecha mejor los recursos de cómputo disponibles en la arquitectura en paralelo. Las implementaciones se han nombrado de la siguiente manera, LUT, Alternative, Opt y Recursive. En la Figura 6.1 se muestran las 4 implementaciones de la estructura de datos en las que se llevó a cabo el experimento.

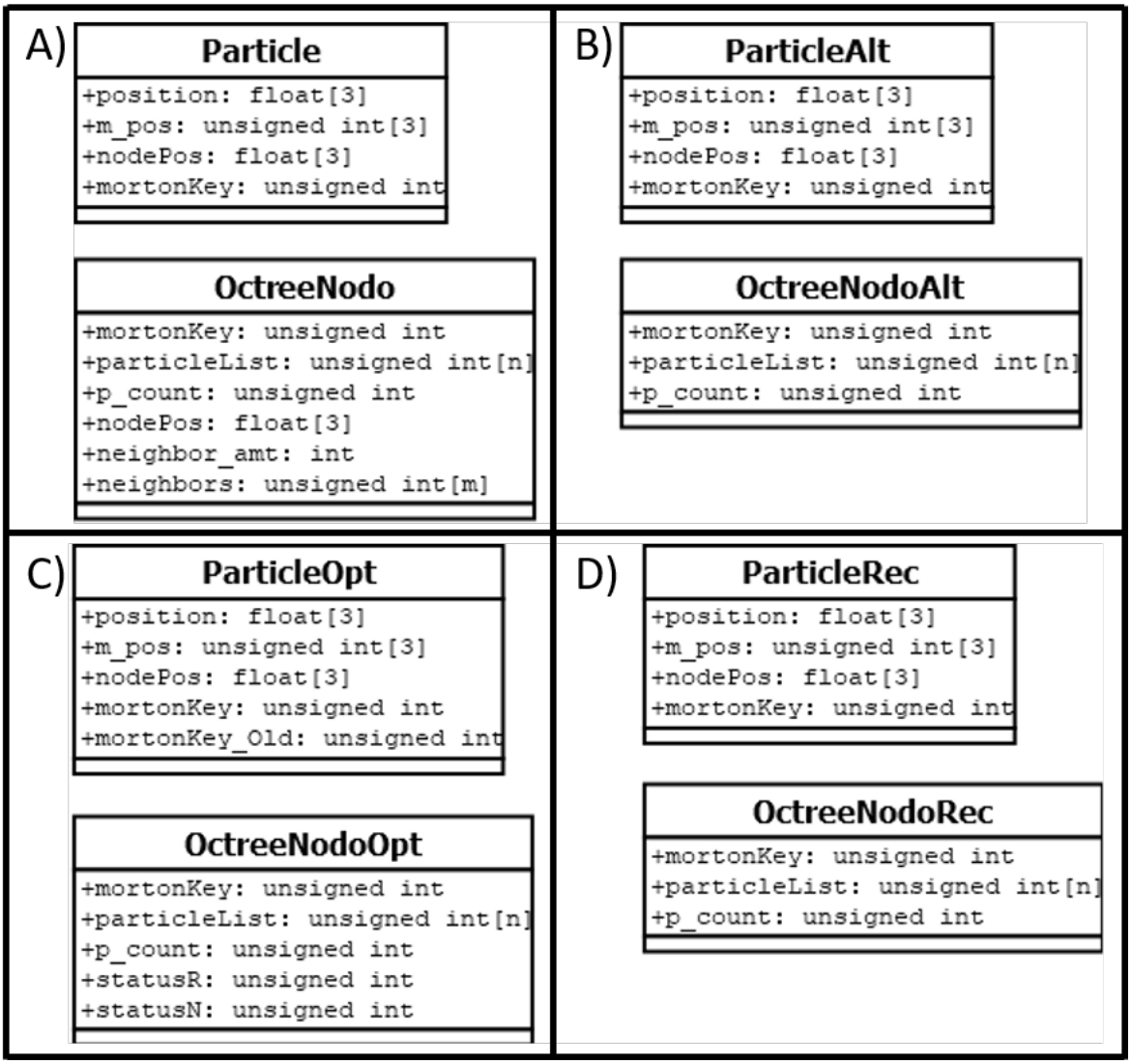

Figura 6.1: Diagramas UML de estructuras de datos implementadas para las pruebas de cálculo de vecindad. A) LUT. B) Alternative. C) Opt. D) Recursive.

Las implementaciones previamente mencionadas tienen además las siguientes especificaciones. El valor m de la variable *neighbors* en la estructura OctreeNodo es de 13 para la implementación usada en este trabajo. El proceso recursivo tiene además

un Octree de navegación lógico, esta estructura se emplea únicamente para navegar y determinar que nodos hijos cuentan con partículas en su interior, esto permite utilizar las capacidades de llamada de kernel recursivo de CUDA, lo cual tiene como finalidad ejecutar conjuntos de hilos a partir de nodos padre que tengan hijos con partículas.

En este experimento se midió el desempeño de un ciclo compuesto de 4 procesos secuenciales, a continuación, se enuncian los procesos y sus diferencias en implementación:

Update Particles. En este proceso se actualizan los valores de las partículas, como son posición y clave Morton. Las implementaciones LUT, Alternative y Recursive son iguales entre sí, mientras que la implementación Opt tiene la diferencia de que guarda la clave Morton anterior y la clave Morton actualizada.

Reset Octree. En este proceso se reinicia cada uno de los nodos que tiene el Octree, para ello es necesario reiniciar los apuntadores a las partículas. Los procesos para cada una de las diferentes implementaciones son casi idénticos. En el caso del proceso Opt se reinician los nodos con ayuda de las claves Morton de las partículas y empleando un semáforo para que solo se reinicie una vez el nodo indicado por una partícula. En el caso del proceso Recursivo se hace uso del Octree de navegación para la ejecución recursiva, esta estructura también se reinicia.

Update Octree. En este proceso se asignan los apuntadores del nodo a cada una de las partículas que tengan la misma clave Morton, además se actualiza el número total de partículas en cada nodo. Los procesos son casi idénticos para cada una de las implementaciones, sin embargo, en el caso del proceso recursivo se actualiza el Octree de navegación.

Get Neighbors. En este proceso se calculan las parejas de partículas, es decir para cada partícula contenida en un nodo se indica su partícula vecina dentro de los nodos adyacentes no vacíos. En la implementación LUT se tiene una lista de nodos vecinos para cada nodo, por lo que solo se usan dichas referencias y se crean los pares de partículas. En la implementación Alternative se calcula la clave Morton de los vecinos durante la ejecución del hilo de cada nodo que tiene partículas. El proceso Opt calcula la vecindad, pero solo ejecuta hilos en los nodos que posean al menos una partícula. La implementación Recursive emplea el Octree de navegación para la ejecución recursiva

de los hilos del proceso, esto se realiza del mismo modo que el proceso Reset Octree.

Una diferencia entre los procesos es en el modo de ejecución de la arquitectura en paralelo. Las implementaciones LUT, Alternative y Recursive se ejecutan bajo un esquema de invocación de hilos en paralelo por cada Nodo, sin embargo. La implementación Recursive reduce el número final de hilos mediante la anulación de algunos de estos utilizando una búsqueda recursiva en donde si un nodo padre no tiene nodos hijos con partículas, no se mandan a ejecutar esa porción de hilos. Por último, la implementación Opt, ejecuta hilos para cada una de las partículas, por lo que se agrega una condición de carrera y un semáforo, el cual evita que se ejecute más de un hilo por nodo.

En la Figura 6.2 se muestra el diagrama de flujo del proceso para el calculo de las vecindades. Este proceso consta de dos partes, una inicialización de datos, la cual no es ejecutada en la GPU, sino en el Procesador CPU, este espacio se denomina huésped (Host). La segunda parte se denomina dispositivo (Device) y se ejecuta en GPU. Los resultados obtenidos solo consideran el proceso en Device ya que el proceso en Host solo corresponde a la inicialización.

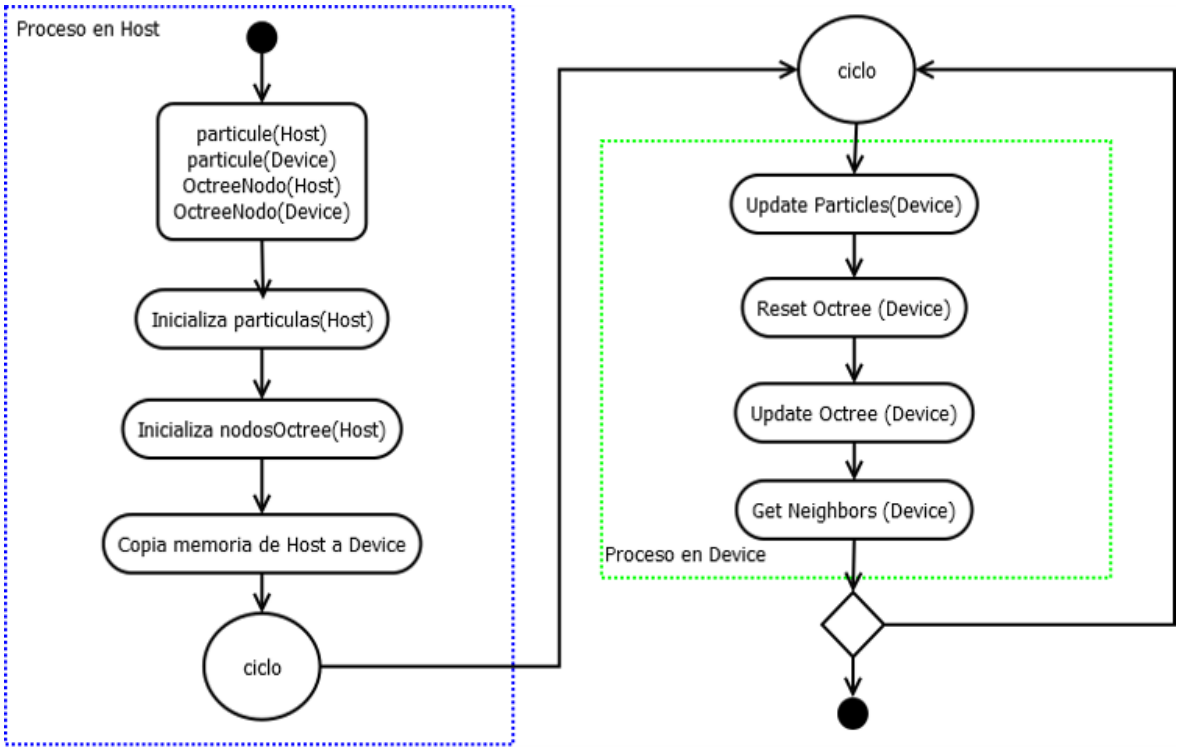

Figura 6.2: Diagrama de flujo del proceso usado para el calculo de vecindad.

Se usaron tres diferentes escenarios para realizar el experimento, en la Figura 6.3 se muestra su representación gráfica. A continuación, se enumeran dichos escenarios y sus características:

A Nube de puntos con distribución aleatoria, consta de un total de 500,000 de partículas.

B Nube de puntos con distribución aleatoria, consta de un total de 1,000,000 de partículas.

C Modelos de ramas y bifurcaciones, creado artificialmente, compuesto de elementos distribuidos de manera uniforme, con un total de 495,839 partículas.

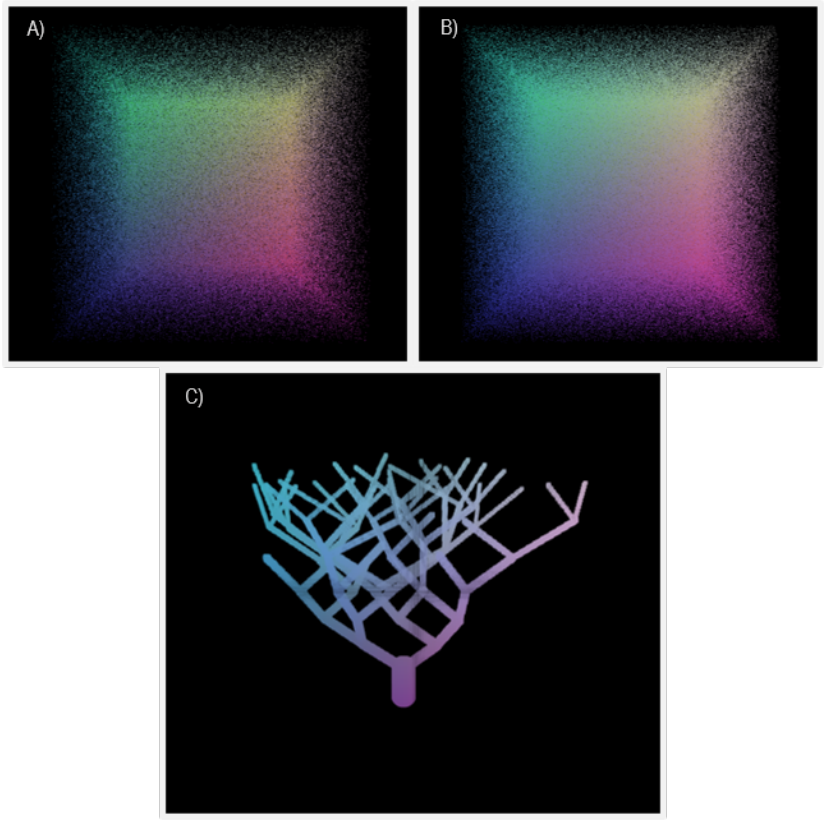

Figura 6.3: Escenarios empleados para realización de experimentos. A) Nube de puntos aleatoria (500,000 partículas). B) Nube de puntos aleatoria (1,000,000 partículas). C) Modelo de ramas y bifurcaciones (495,839 partículas).

Las Tablas 6.1, 6.2 y 6.3 contienen los resultados de desempeño medidos para cada uno de los escenarios. Cada tabla contiene los resultados para tres diferentes configuraciones de Octree con niveles de profundidad 7, 8 y 9.

A	LEVEL 7				LEVEL 8				LEVEL 9			
Event	LUT	Alternative	Opt	Recursive	LUT	Alternative	Opt	Recursive	LUT	Alternative	Opt	Recursive
Total Time [ms]	6032	3256	4507	3677	9710	7283	4657	9564	---	41889	4726	62569
Loop Time [ms]	6.030	3.255	4.505	3.676	9.701	7.277	4.654	9.560	---	41.885	4.724	62.562
Update Particles [ms]	0.431	0.431	0.581	0.337	0.430	0.431	0.583	0.337	---	0.430	0.583	0.342
Reset Octree [ms]	0.994	0.920	1.094	1.144	2.895	2.816	1.136	3.290	---	20.454	1.170	24.132
Update Octree [ms]	1.120	1.118	1.121	1.202	1.120	1.121	1.124	1.191	---	1.119	1.130	1.197
Get Neighbors [ms]	3.124	0.455	1.371	0.672	4.901	2.562	1.449	4.405	---	19.455	1.462	36.483
Nodes [MB]	664	528	544	528	1728	640	768	640	---	5120	6144	5120
Particles [MB]	19	19	21	19	19	19	21	19	---	19	21	19
Neighbors [Pairs]	1584803	1584803	1584803	1584803	200160	200160	200160	200160	---	25068	25068	25068

Cuadro 6.1: Comparación de desempeño entre implementaciones en escenario A.

B	LEVEL 7				LEVEL 8				LEVEL 9			
Event	LUT	Alternative	Opt	Recursive	LUT	Alternative	Opt	Recursive	LUT	Alternative	Opt	Recursive
Total Time [ms]	12038	5175	8682	5452	12909	9202	8877	11072	---	43982	9028	65128
Loop Time [ms]	12.036	5.172	8.678	5.450	12.907	9.196	8.874	11.072	---	43.978	9.023	65.124
Update Particles [ms]	0.840	0.845	1.155	0.650	0.841	0.843	1.152	0.645	---	0.842	1.150	0.654
Reset Octree [ms]	1.332	1.229	2.116	1.437	3.014	2.889	2.265	3.327	---	20.996	2.321	25.247
Update Octree [ms]	2.240	2.234	2.253	2.365	2.242	2.241	2.255	2.346	---	2.245	2.264	2.357
Get Neighbors [ms]	7.209	0.490	2.592	0.675	6.422	2.877	2.865	4.402	---	19.487	2.913	36.437
Nodes [MB]	664	528	544	528	1728	640	768	640	---	5120	6144	5120
Particles [MB]	38	38	42	38	38	38	42	38	---	38	42	38
Neighbors [Pairs]	6337674	6337674	6337674	6337674	799618	799618	799618	799618	---	100316	100316	100316

Cuadro 6.2: Comparación de desempeño entre implementaciones en escenario B.

C	LEVEL 7				LEVEL 8				LEVEL 9			
Event	LUT	Alternative	Opt	Recursive	LUT	Alternative	Opt	Recursive	LUT	Alternative	Opt	Recursive
Total Time [ms]	2442	1373	972	1073	6032	5694	1132	1795	---	40152	2467	8890
Loop Time [ms]	2.442	1.371	0.970	1.072	6.032	5.693	1.130	1.795	---	40.151	2.463	8.889
Update Particles [ms]	0.329	0.330	0.420	0.341	0.329	0.333	0.434	0.343	---	0.334	0.423	0.334
Reset Octree [ms]	0.386	0.330	0.114	0.149	2.472	2.467	0.144	0.467	---	19.533	0.580	3.152
Update Octree [ms]	0.134	0.132	0.134	0.117	0.163	0.152	0.156	0.156	---	0.526	0.593	0.546
Get Neighbors [ms]	1.350	0.319	0.114	0.169	2.759	2.432	0.185	0.611	---	19.396	0.634	4.559
Nodes [MB]	664	528	544	528	1728	640	768	640	---	5120	6144	5120
Particles [MB]	19	19	21	19	19	19	21	19	---	19	21	19
Neighbors [Pairs}	173971250	173971250	173971250	173971250	200160	200160	200160	200160	---	25068	25068	25068

Cuadro 6.3: Comparación de desempeño entre implementaciones en escenario C.

Se analizó con mayor detenimiento el proceso *Get Neighbors*, dado que es en este proceso donde se emplea el método propuesto.

Las Figuras 6.4, 6.5 y 6.6 muestran las gráficas con el detalle de desempeño de los diferentes métodos en cada uno de los escenarios y configuraciones de Octree para el calculo de *Get Neighbors*

El comportamiento observado para el proceso *Get Neighbors*, hace notar que al

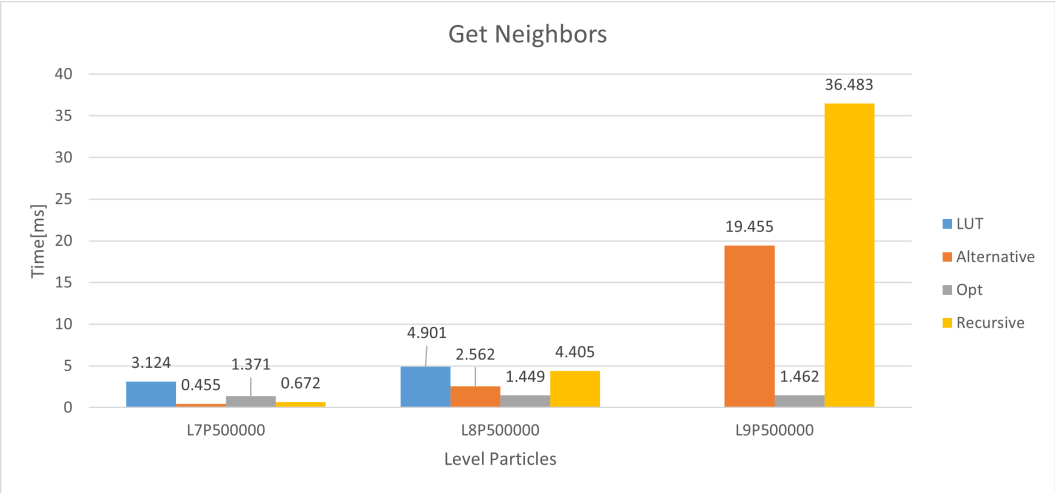

Figura 6.4: Gráfica de rendimiento de implementaciones LUT. Alt, Rec y Opt para escenario (A) empleando Octree con nivel de profundidad 7,8 y 9.

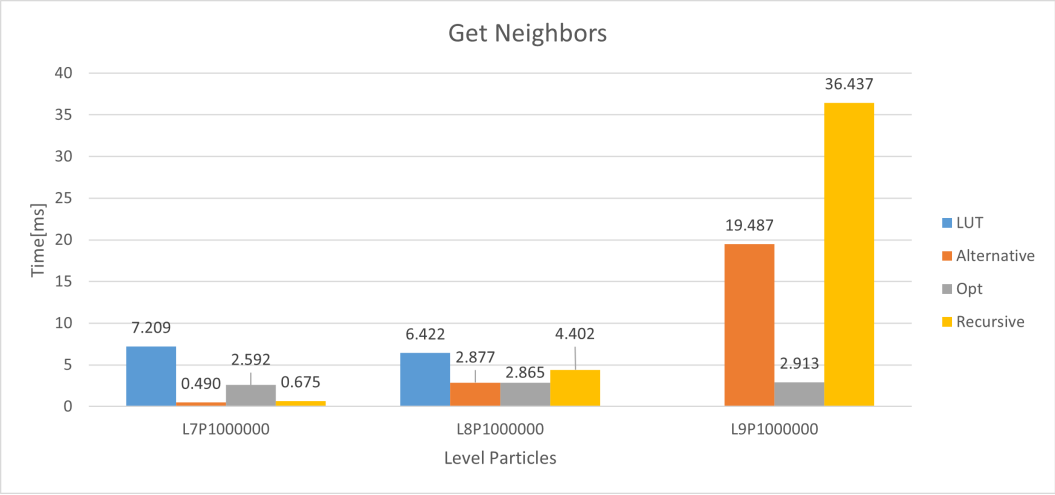

Figura 6.5: Gráfica de rendimiento de implementaciones LUT. Alt, Rec y Opt para escenario (B) empleando Octree con nivel de profundidad 7,8 y 9.

incrementar el nivel de profundidad el proceso es más costoso en tiempo, esto se observa principalmente en las implementaciones LUT, Alternative y Recursive, mientras que la implementación OPT se mantiene constante. Este compartimento es debido a que el número de hilos ejecutados para la implementación Opt depende de la cantidad de partículas y los nodos ocupados, mientras que en las otras implementaciones solo depende de la profundidad.

Al analizar el comportamiento considerando la cantidad de partículas, se observa

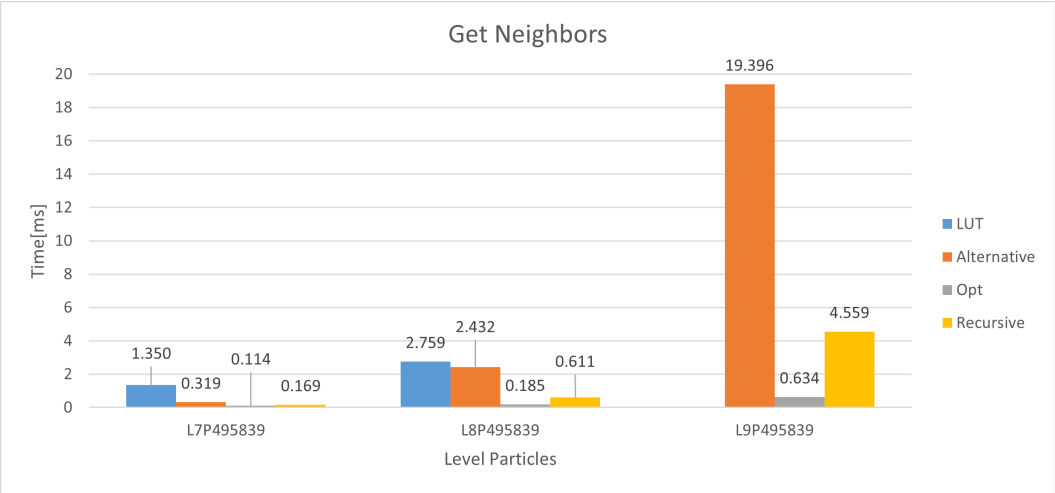

Figura 6.6: Gráfica de rendimiento de implementaciones LUT. Alt, Rec y Opt para escenario (C) empleando Octree con nivel de profundidad 7,8 y 9.

que existe una relación lineal entre el desempeño y el número de partículas en las implementaciones LUT, Alternative y Opt, sin embargo, la implementación Recursive depende más de la distribución de las partículas y no de la cantidad; esta implementación se beneficia de distribuciones más aglomeradas que de las que muestran mayor dispersión en todo el espacio que comprende el Octree.

Para analizar con mayor detalle y determinar la razón del comportamiento observado en cada una de las implementaciones, se decidió medir el desempeño mediante el uso de la herramienta Nvidia Nsight, la cual está disponible para los ambientes de desarrollo de Microsoft Visual Studio. Mediante esta herramienta se obtuvieron los reportes de numero de operaciones y uso de memoria en GPU del proceso *Get Neighbors*. En la Figura 6.7 y Tabla 6.4 se muestran los resultados de ejecución de Operaciones de enteros por segundo (IOPS por las siglas en inglés de Integer Operations per Second); en la Figura 6.8 y Tabla 6.5 se muestran los resultados de uso y acceso a memoria en GPU.

Al analizar los resultados se observa que la implementación Opt realiza el menor número de IOPS; por otro lado, la implementación Recursive es la que realiza el mayor numero de IOPS, sin embargo, los resultados de desempeño obtenidos sugieren que la implementación Recursive no es la menos eficiente, por lo que es necesario complementar el análisis junto con la información de uso y acceso de memoria en GPU.

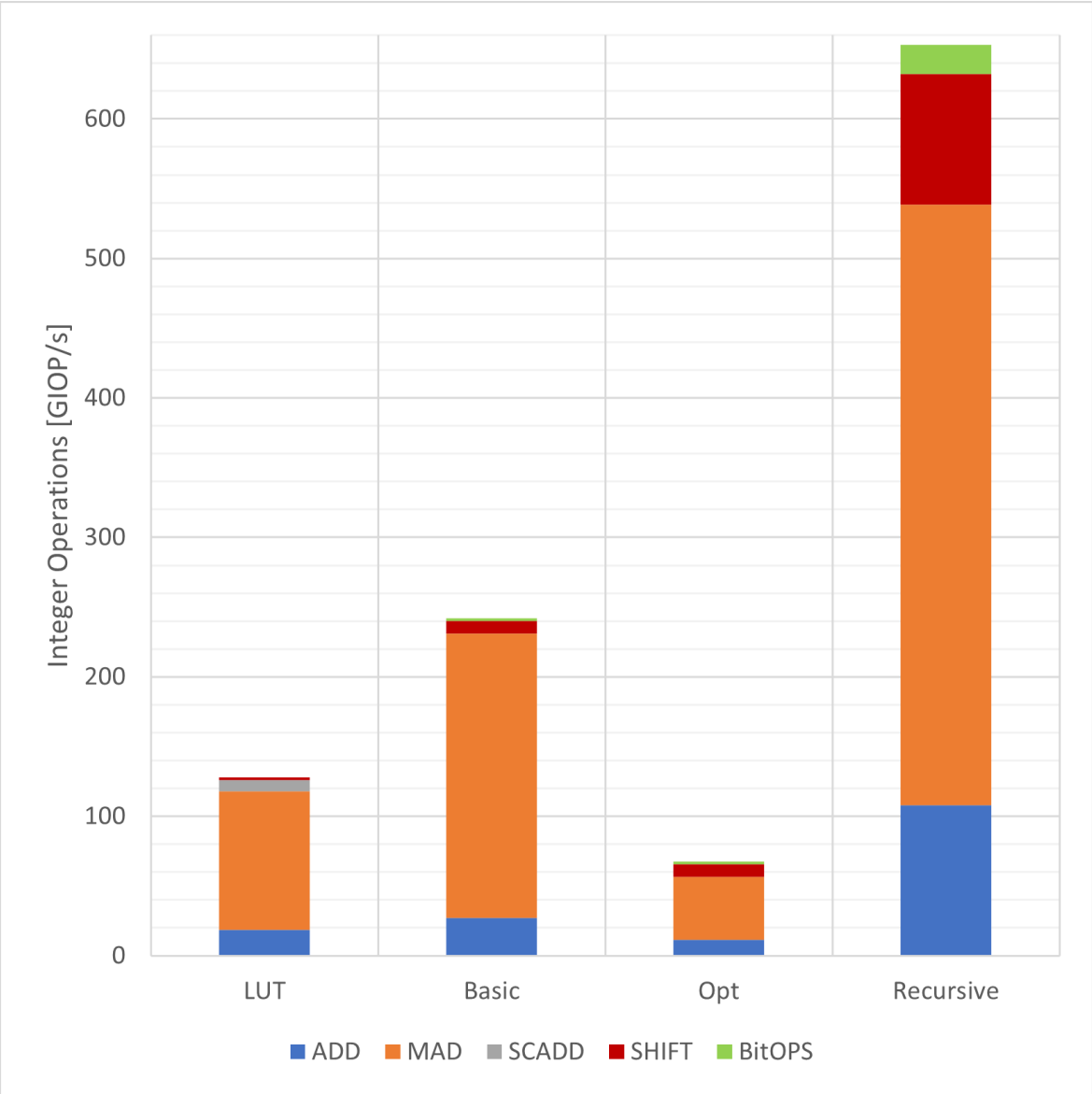

Figura 6.7: Diagrama de memoria usada por implementaciones LUT, Alternative, Recursive y Opt. Diagramas generados mediante Nvidia Nsight para Microsoft Visual Studio.

Cuando se analizan los resultados obtenidos del uso y acceso de memoria, se observa que la implementación Opt es la que menos memoria Global emplea, aunque ocupa más memoria Global Atomic que las otras implementaciones, es notorio que hay una relación entre uso de memoria global y desempeño. Esto es fácilmente observable al comparar los resultados de desempeño de las diferentes implementaciones del proceso *Get Neighbors* del escenario B con un Octree a nivel 8; se observa que la implementación LUT tiene el peor desempeño y es la que más memoria Global utiliza, mientras que

Integer Operations	LUT	Alternative	Opt	Recursive
ADD [GIOP/s]	18.6596	27.3022	11.3442	107.7366
MAD [GIOP/s]	99.3522	203.8334	45.3773	430.8697
SCADD [GIOP/s]	7.8279	0	0	0
SHIFT [GIOP/s]	1.957	8.9524	8.94	93.7242
BitOPS [GIOP/s]	0	2.0806	1.9979	20.9648

Cuadro 6.4: Uso de memoria en GPU por parte de las implementaciones LUT, Alternative, Recursive Y Opt.

Memory	LUT	Alternative	Opt	Recursive
Global [MB]	1075.2	519.46	89.77	530.69
Local [MB]	0	0	0	263.87
Global Atomics [MB]	0	0	61.04	5.34
Shared Atomics [MB]	0	0	0	0
Shared [MB]	0	0	0	0

Cuadro 6.5: Uso de memoria en GPU por parte de las implementaciones LUT, Alternative, Recursive Y Opt.

la implementación Opt tiene el mejor desempeño y requiere la menor cantidad de memoria Global; por otro lado las implementaciones Alternative y Recursive están casi empatadas en su uso de memoria global y presentan un desempeño muy similar.

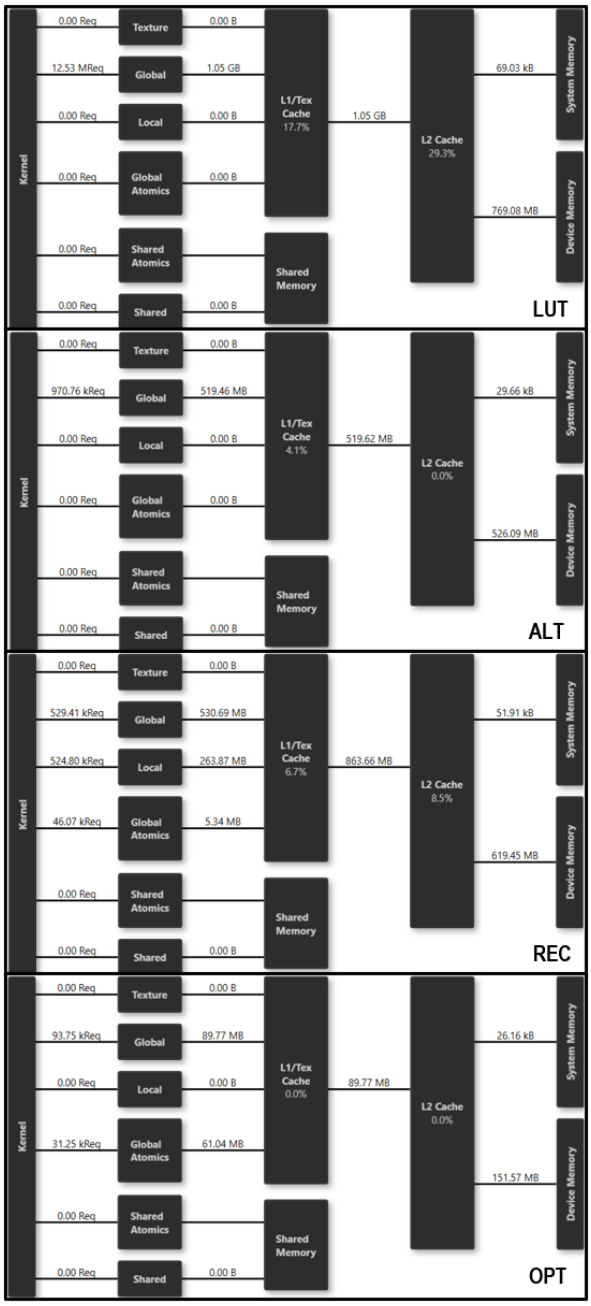

Figura 6.8: Diagrama de memoria usada por implementaciones LUT, Alternative, Recursive y Opt. Diagramas generados mediante Nvidia Nsight para Microsoft Visual Studio.

6.2. Exp 2. Cálculo de vecindad con ordenamiento por clave Morton

Siguiendo con las conclusiones de la primera prueba, se puede observar que el método Opt es el que presenta los mejores resultados. En está prueba se compara la implementación Opt contra una implementación similar que además incorpora ordenamiento por clave Morton, está implementación se denomina Sorted.

Los experimentos realizados comprenden los mismos tres escenarios usados en Exp 1. De manera adicional se ejecutaron los experimentos empleando 3 diferentes equipos de cómputo, dichos equipos cuentan con procesadores y memoria similares, sin embargo, las tarjetas de vídeo son diferentes. A continuación, se indican las tarjetas de vídeo con las que cada equipo cuenta:

- GPU_1: Nvidia GeForce GTX 1070 8GB.
- GPU_2: Nvidia GeForce GTX 1080 8GB.
- GPU_3: Nvidia GeForce GTX 1080Ti 11GB.

En las Tablas 6.6, 6.7 y 6.8 se muestran los resultados de desempeño medidos en las implementaciones Opt y Sorted usando Otrees con nivel máximo de profundidad 7, 8 y 9 respectivamente.

Los resultados para Opt muestran los tiempos para el calculo de *Get Neighbors* y para la implementación Sorted se muestra tanto el desempeño del método de cálculo de vecindad (A) y ordenamiento por clave Morton (B), así como la suma de ambos. Con el fin de observar esto más claramente, se ilustra esta comparativa en las Figuras 6.9, 6.10 y 6.11.

Al analizar los resultados se observa que la implementación SORTED tiene mejor desempeño si solo se considera el proceso de búsqueda de vecinos (UpdateNeighborhoodOpt y UpdateNeighborhoodSorted), sin embargo, al considerar el proceso de Ordenamiento por clave Morton (ThrustSort), se observa que el proceso de ordenamiento reduce su desempeño conforme el nivel de profundidad de la clave Morton se incrementa.

	LEVEL 7				
GPU	OPT [ms]	SORTED (A+B) [ms]	A [ms]	B [ms]	
GPU_1	178.553	88.282	83.957	4.325	C
GPU_2	232.831	87.065	83.547	3.519	
GPU_3	153.094	67.867	64.481	3.386	
GPU_1	9.410	6.832	2.629	4.203	A
GPU_2	12.221	8.459	3.424	5.035	
GPU_3	7.955	5.744	2.225	3.519	
GPU_1	26.401	12.985	5.333	7.652	B
GPU_2	38.155	16.796	7.182	9.614	
GPU_3	22.702	11.992	4.750	7.242	

Cuadro 6.6: Resultados de desempeño de implementaciones OPT y SORTED empleando un Octree con nivel máximo de profundidad 7, usando los 3 escenarios de Exp 1.

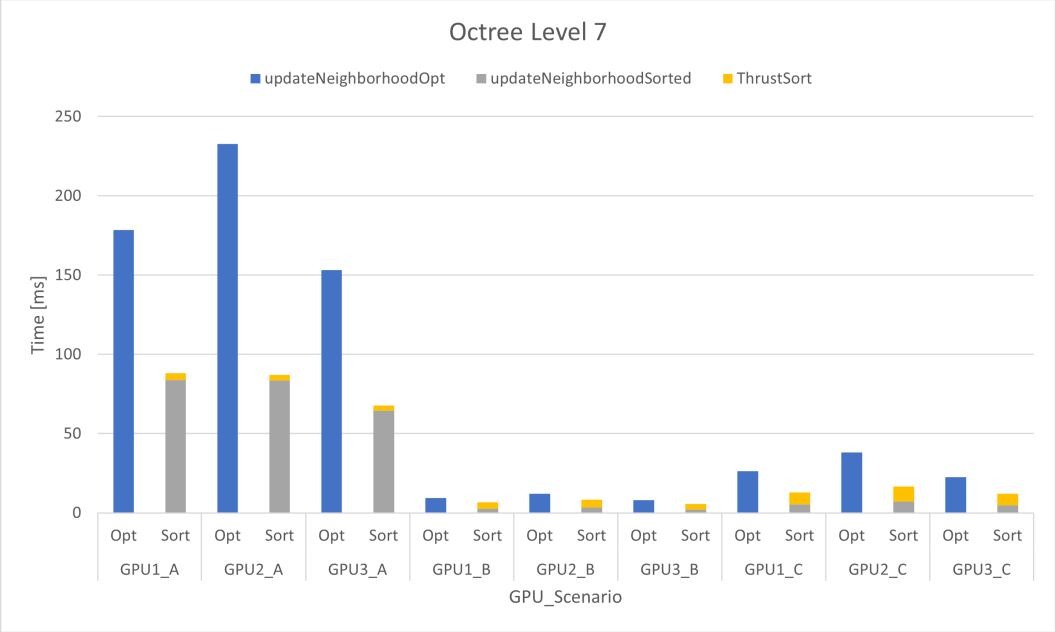

Figura 6.9: Gráfica de comparativa de desempeño entre implementaciones OPT y SORTED con un Octree con nivel máximo de profundidad 7. Se usaron 3 escenarios y 3 GPUs diferentes.

Algo que se debe mencionar de está implementación es que el proceso de cálculo de vecindad fue ajustado para que se realizaran cálculos de distancias entre vecinos, estos son cálculos similares a los encontrados en los métodos de SPH, por lo que en este experimento también es posible observar como el incremento en el nivel máximo de profundidad del Octree, mejora el desempeño del cálculo de vecindad.

	LEVEL 8				
GPU	OPT [ms]	SORTED (A+B) [ms]	A [ms]	B [ms]	
GPU_1	19.575	12.503	8.454	4.048	C
GPU_2	37.535	12.440	8.665	3.775	
GPU_3	23.176	9.332	6.012	3.320	
GPU_1	6.387	8.305	4.172	4.133	A
GPU_2	8.822	10.661	5.719	4.942	
GPU_3	5.427	7.255	3.690	3.564	
GPU_1	13.765	15.103	7.265	7.838	B
GPU_2	18.747	20.809	10.147	10.662	
GPU_3	11.633	13.784	6.451	7.333	

Cuadro 6.7: Resultados de desempeño de implementaciones OPT y SORTED empleando un Octree con nivel máximo de profundidad 8, usando los 3 escenarios de Exp 1.

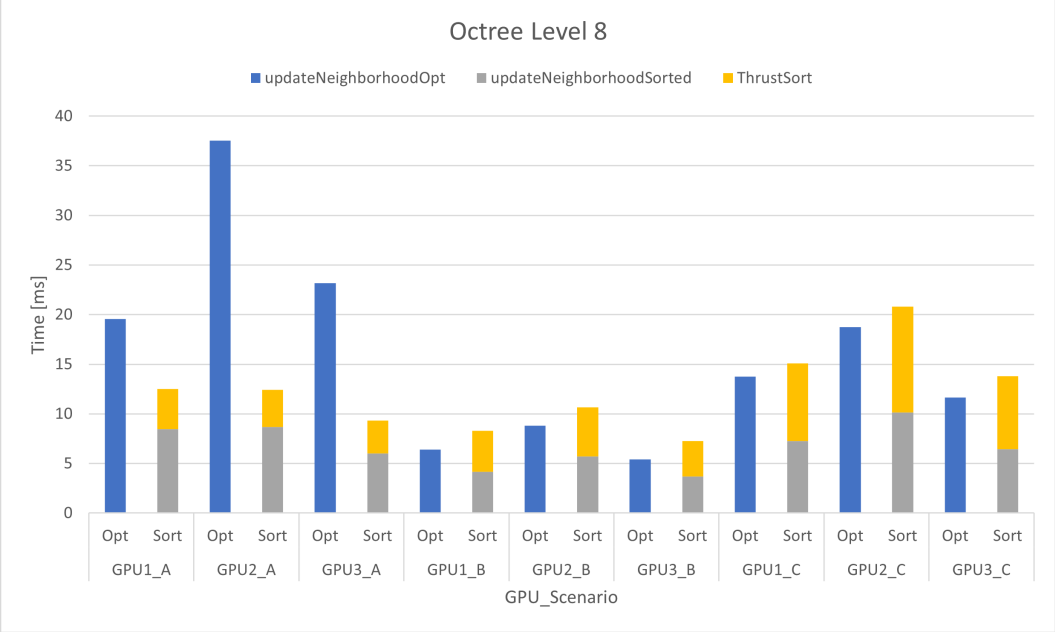

Figura 6.10: Gráfica de comparativa de desempeño entre implementaciones OPT y SORTED con un Octree con nivel máximo de profundidad 8. Se usaron 3 escenarios y 3 GPUs diferentes.

	LEVEL 9				
GPU	OPT [ms]	SORTED (A+B) [ms]	A [ms]	B [ms]	
GPU_1	5.824	6.763	2.746	4.017	C
GPU_2	9.064	8.702	4.581	4.121	
GPU_3	5.913	5.980	2.866	3.113	
GPU_1	6.110	9.930	5.484	4.446	A
GPU_2	7.958	12.824	7.568	5.256	
GPU_3	5.117	8.300	4.673	3.627	
GPU_1	12.264	19.106	10.222	8.884	B
GPU_2	16.153	24.073	13.745	10.327	
GPU_3	10.349	15.708	8.830	6.878	

Cuadro 6.8: Resultados de desempeño de implementaciones OPT y SORTED empleando un Octree con nivel máximo de profundidad 8, usando los 3 escenarios de Exp 1.

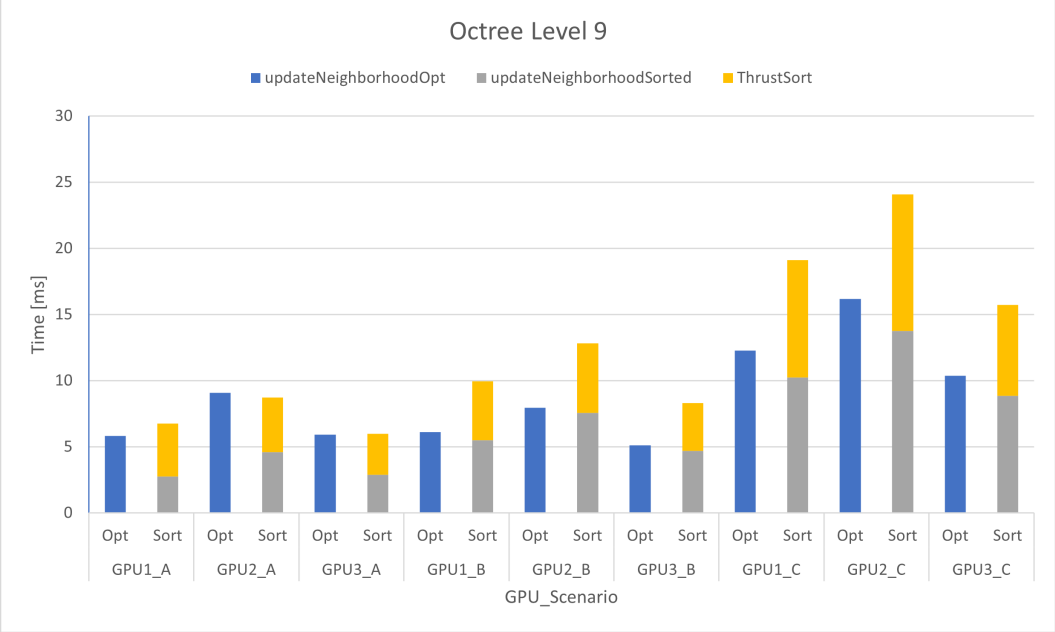

Figura 6.11: Gráfica de comparativa de desempeño entre implementaciones OPT y SORTED con un Octree con nivel máximo de profundidad 9. Se usaron 3 escenarios y 3 GPUs diferentes.

6.3. Exp 3. Simulación de fluidos

Los métodos presentados en este trabajo se enfocan en mejorar los tiempos de ejecución de soluciones basadas en métodos de partículas suavizadas, y el enfoque buscado es mejorar el proceso de calculo de vecindad proponiendo un método de Octree dinámico que hace uso de las capacidades de las GPUs.

En el Exp. 3 busca mostrar que el método implementado puede ser usado para realizar cálculos numéricos al simular el comportamiento de flujos por medio de una implementación de SPH.

En esta sección el trabajo se enfocara en mostrar las capacidades del método para obtener soluciones correctas en configuraciones de las cuales podemos calcular soluciones analíticas, no se presentan pruebas de tiempos de ejecución ya que estas se harán en configuraciones mas complejas que requieran un numero mayor de partículas.

La primera prueba muestra la simulación de un flujo laminar que se mueve en una sola dirección entre dos placas paralelas. La solución analítica de esta configuración se obtiene con la ecuación del flujo de Poiseuille en un plano (6.1).

$$v = \frac{1}{2\eta}\frac{\Delta P}{L}(y^2 - yh_y). \tag{6.1}$$

Donde ΔP es la diferencia de presión, η la viscosidad, la separación entre las placas es h_y y L es el largo de las placas.

La velocidad dentro del tubo tiene un perfil parabólico, por lo que la velocidad del fluido es mayor cuando se encuentra al centro del tubo y conforme se acerca a la pared del tubo tiende a cero, esto se muestra en el diagrama de la Figura 6.12.

En la Tabla 6.9 se muestran los parámetros de cada una de las pruebas realizadas.

En las Figuras 6.13, 6.14 y 6.15, se muestran las comparaciones entre el resultado analítico y la simulación del flujo con la implementación de SPH de las ecuaciones de Navier-Stokes.

Como se puede observar se obtiene la misma solución con el método de simulación

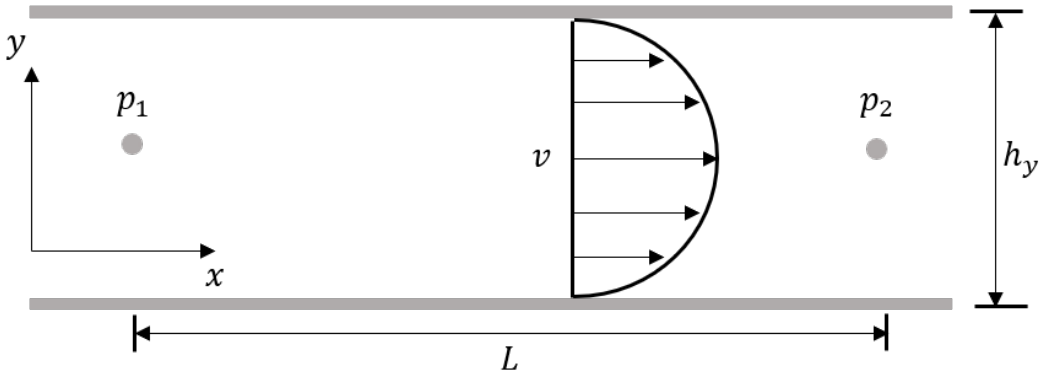

Figura 6.12: Diagrama con perfil de velocidades parabólico de fluido laminar dentro de un tubo.

	ΔP [Pa]	**η [Pa ·s]**	**L [m]**	**h [m]**
Prueba 1	-0.057	0.00089	0.00435	0.001653
Prueba 2	-0.570	0.00089	0.00435	0.001653
Prueba 3	-5.700	0.00089	0.00435	0.001653

Cuadro 6.9: Parámetros empleados para pruebas de flujo laminar en tubo usando ecuación de Poiseuille.

implementado en el Octree presentado en este trabajo que el calculado analíticamente con la ecuación de Poiseuille.

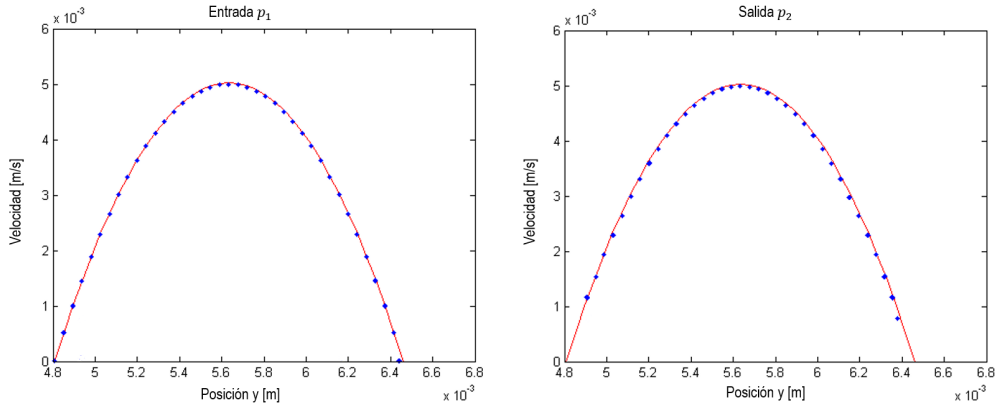

Figura 6.13: Prueba 1. Resultado de simulación (puntos) y analítico (línea) de fluido laminar.

La Figura 6.16 muestra el resultado de la simulación del flujo entre las dos barras paralelas

Esta prueba se puede extender a el flujo dentro de un cilindro, para la cuál también

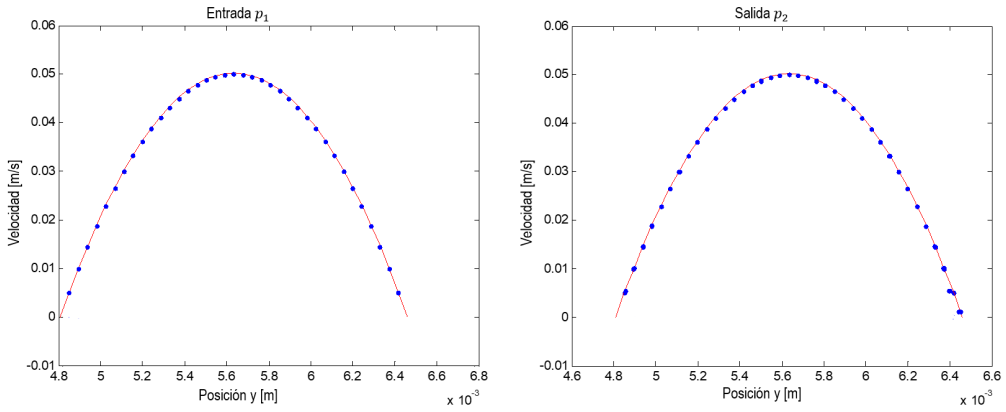

Figura 6.14: Prueba 2. Resultado de simulación (puntos) y analítico (línea) de fluido laminar.

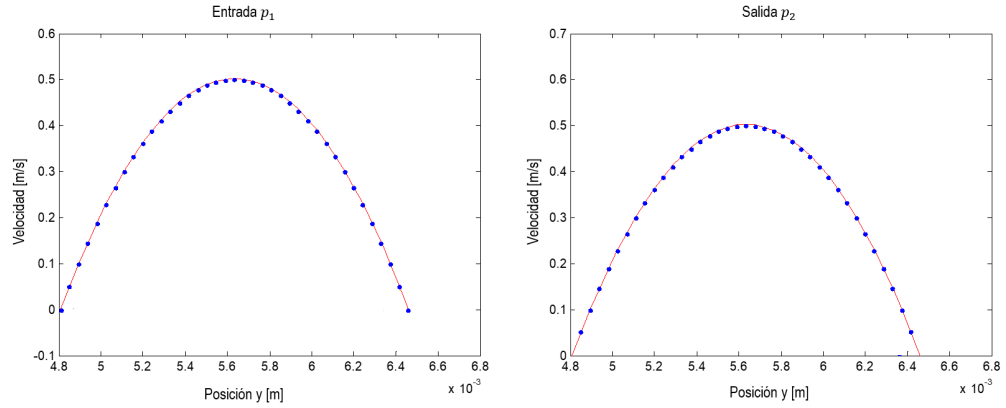

Figura 6.15: Prueba 3. Resultado de simulación (puntos) y analítico (línea) del flujo laminar.

se tiene una solución analítica dada por la ecuación de Hagen–Poiseuille. Este flujo también se da debido a una diferencia de presión entre la entrada y la salida.

En la Figura 6.17 se muestra el corte transversal en el eje XY del cilindro discreto de donde se obtendrán las condiciones de frontera y las partículas que definen a la pared del cilindro. En las pruebas del flujo dentro de un cilindro se muestra el impacto de la resolución del sistema de partículas en el resultado de la simulación.

La Figura 6.18 muestra el resultado de la simulación del fluido a la entrada del cilindro dónde se puede observar el cambio de velocidad con respecto al camino recorrido y algunos de los efectos de la discretización. El flujo cumple el comportamiento de forma de paraboloide tendiendo a velocidad cero en las fronteras.

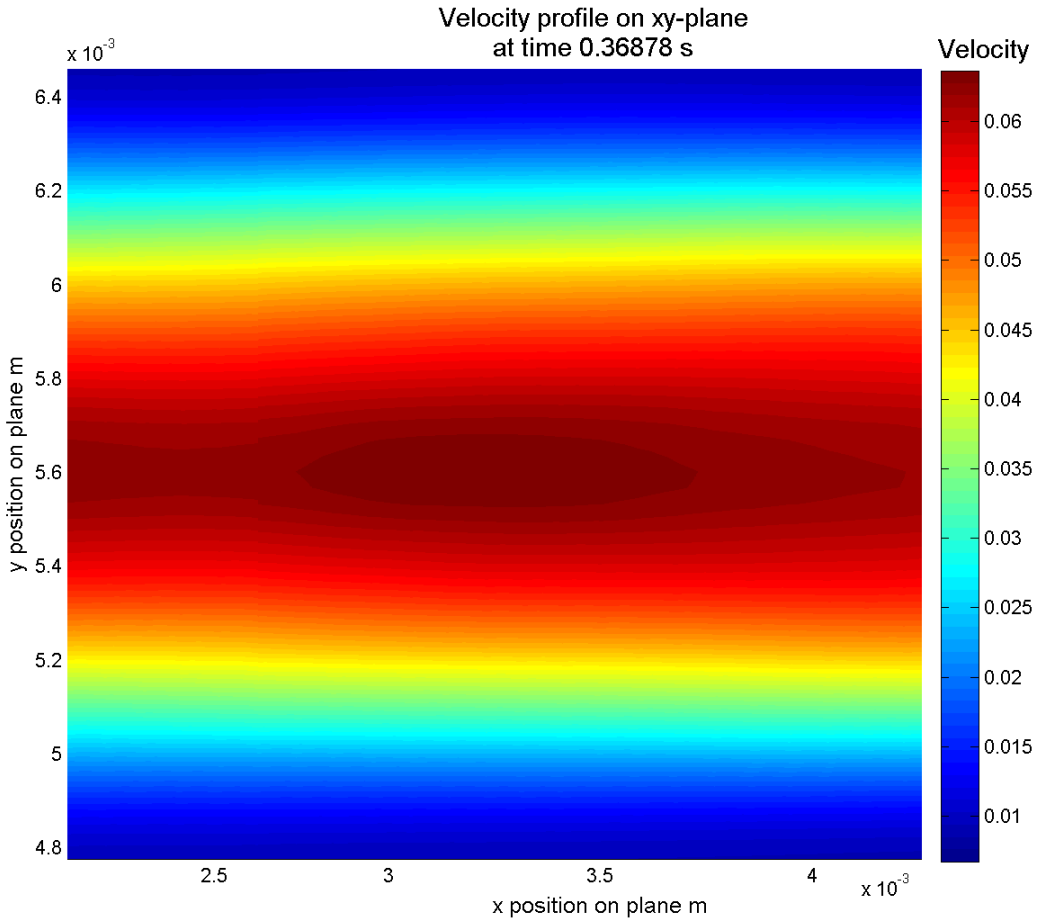

Figura 6.16: Prueba 3. Resultado de simulación del flujo laminar.

En la Figura 6.19 se muestra el flujo en diferentes cortes transversales donde es mas claro el comportamiento paraboloide del fluido y el impacto que tienen las fronteras discretizadas en la simulación.

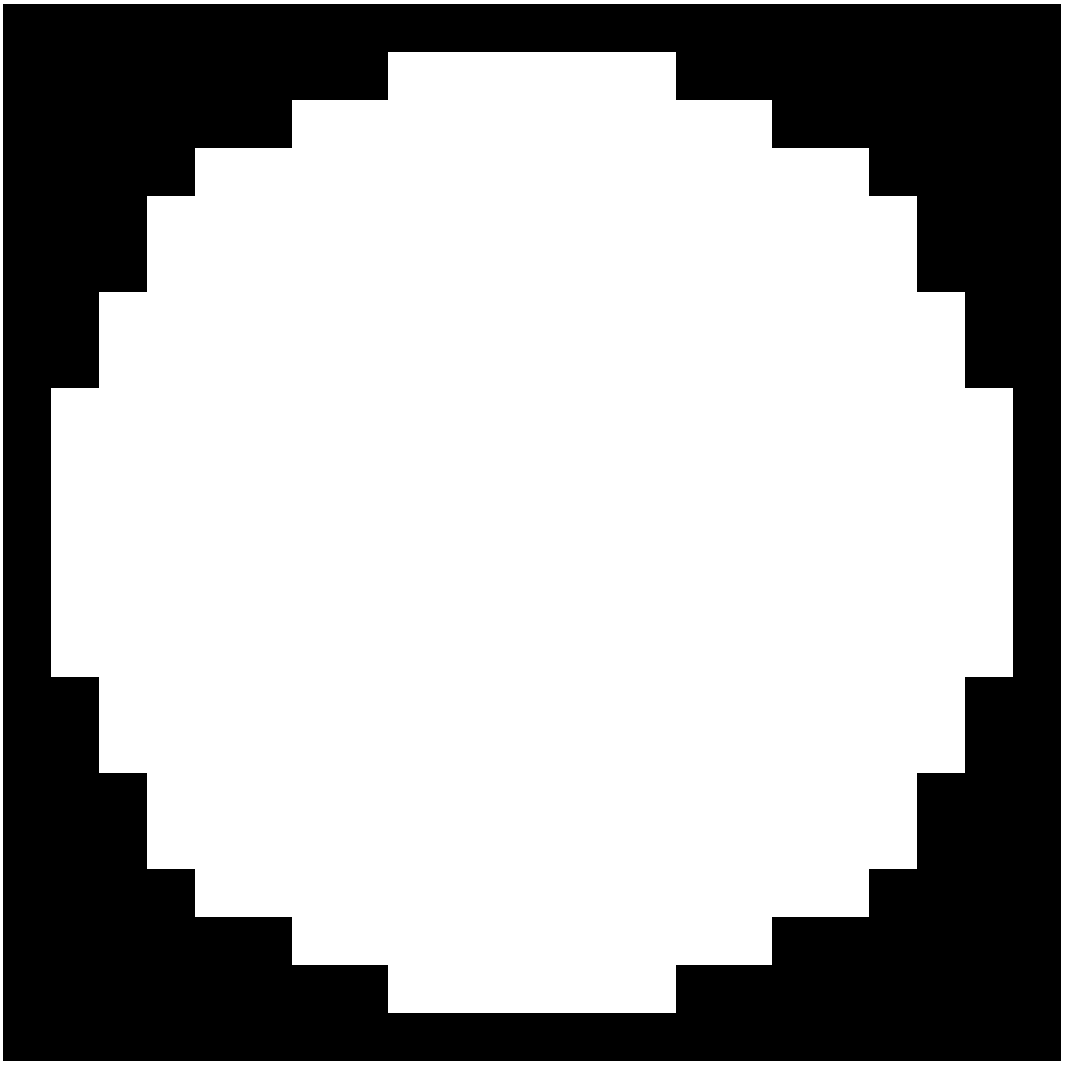

Figura 6.17: Corte transversal de cilindro, se observan errores en la aproximación de la forma debido a la discretización en voxeles (píxeles).

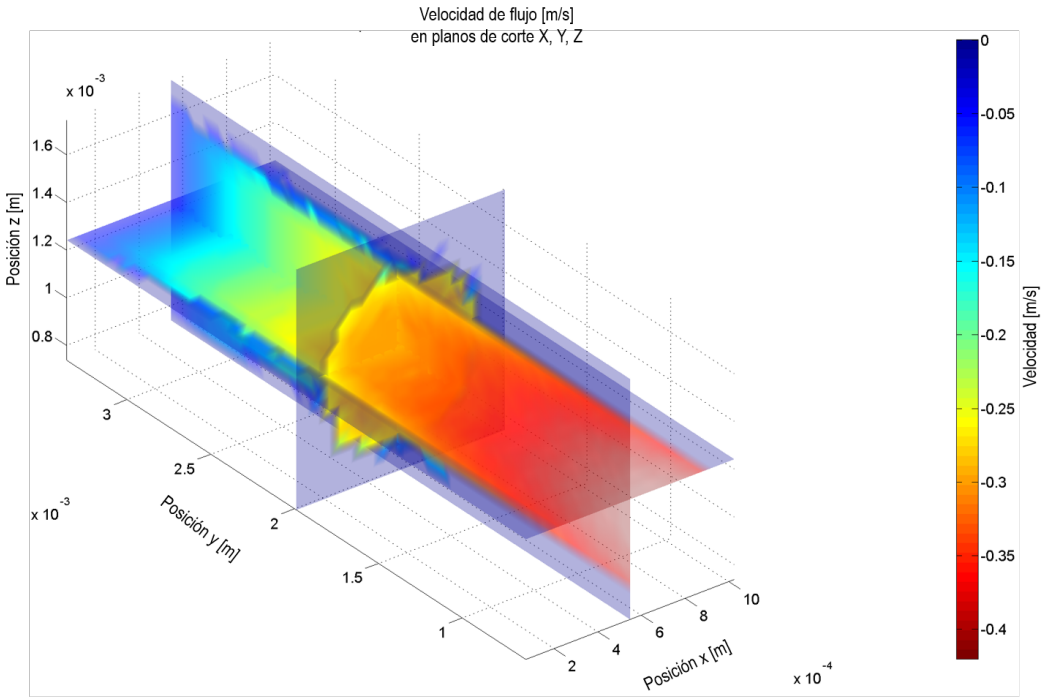

Figura 6.18: Resultado de simulación de flujo donde se gráfica la velocidad promedio de las partículas dentro de un cilindro, se toma como referencia los planos centrales del cilindro.

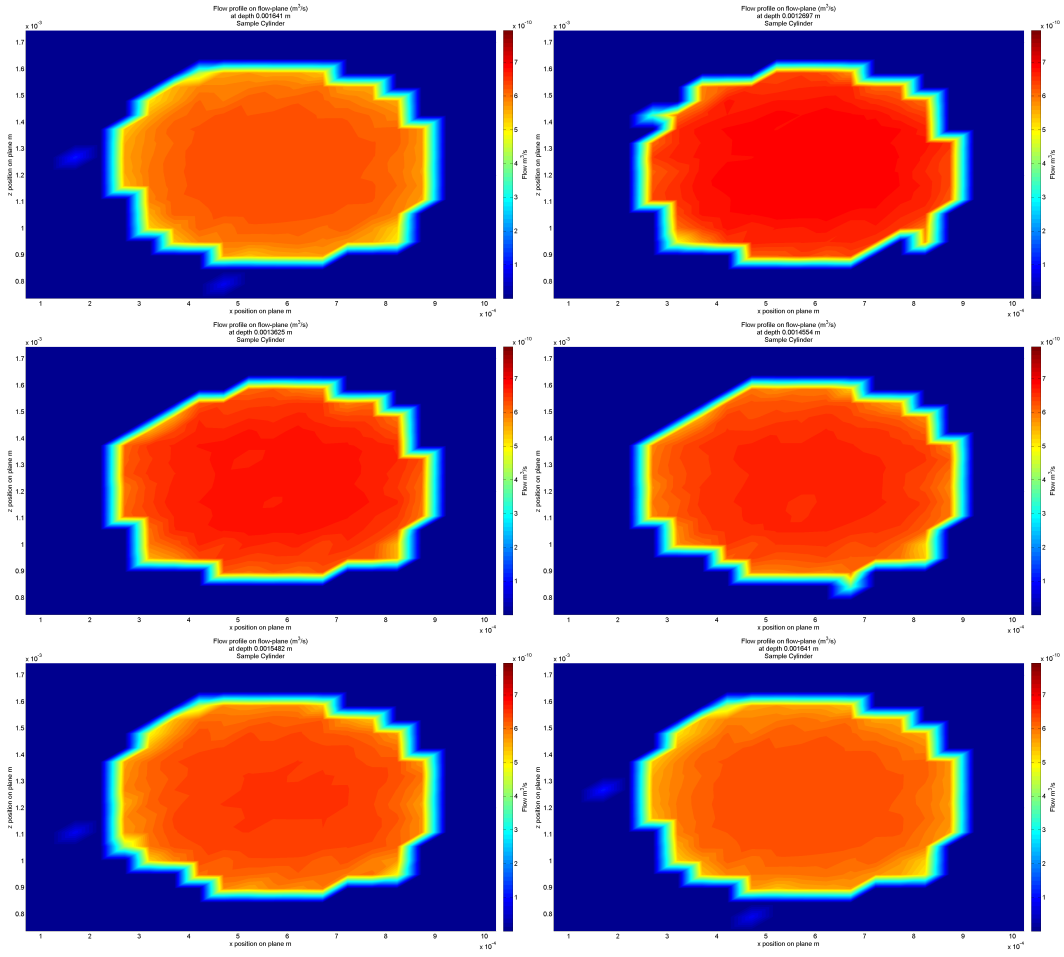

Figura 6.19: Velocidad de partículas en cilindro, los resultados muestran concordancia con la solución analítica.

6.4. Pruebas de velocidad bajo diferentes condiciones

Las pruebas finales de las capacidades del método propuesto tienen que ver con la velocidad de computo de la simulación de sistemas de partículas en diversas configuraciones y con diferente numero de partículas usadas.

6.4.1. Exp 4. Simulación con implementación Opt

En este experimento se implementó un motor de simulación usando el método de SPH mencionado en los capítulos anteriores, así como el método de cálculo de vecindades propuesto en este trabajo. La implementación para el cálculo de vecindad que emplea implementación Opt.

Para este experimento se utilizó un equipo de cómputo con las siguientes especificaciones:

- CPU: Intel i7-7700HQ.
- GPU: Nvidia GeForce GTX 1070 8GB.
- Memoria: 16GB RAM DDR4.

El algoritmo de simulación de fluidos implementado para este experimento consta de los procesos que se describen a continuación:

updateParticle (UP). Integración de posición de cada partícula con base a suma de fuerzas internas y externas. Actualización de propiedades y clave Morton asociada a cada partícula.

resetNodeParticles (RNP). Reinicio de Octree, esto permite preparar la estructura de datos para una posterior actualización de partículas asociadas a cada nodo.

updateOctreeParticles (UOP). Asignación de partículas a su respectivo nodo mediante un apuntador que emplea su clave Morton.

updateNeighborhoodDensity (UND). Cálculo de densidad para las partículas empleando la vecindad obtenida mediante el método de cálculo de vecindad propuesto en este trabajo.

updateParticleRho (UPR). Paso final para el cálculo de densidad como propiedad de campo, en este proceso se integra el valor de aportación propia de la densidad para cada partícula. Para simplificar este proceso este paso se separó del proceso anterior.

updateNeighborhoodForce (UNF). Cálculo de fuerzas internas para las partículas empleando la vecindad obtenida mediante la búsqueda de vecinos anteriormente propuesta.

updateParticleEF (UPEF). Suma de las fuerzas externas que actúan en las partículas, para esté experimento solo es la fuerza debido a la gravedad.

Loop time. Tiempo total del ciclo de procesos antes mencionados.

El experimento se realizó empleando 5 nubes de partículas similares al escenario A y B del experimento 1 y 2, sin embargo, el número de partículas en este experimento fueron de 100,000 a 500,000 partículas, variando en intervalos de 100,000 partículas entre cada escenario. La simulación inicia con las partículas suspendidas dentro de una caja virtual, las cual proporciona un límite artificial que las contiene; estas partículas solo son afectadas por la fuerza externa debida a la gravedad y conforme avanza la simulación se depositan en el fondo de la caja virtual. Considerando un intervalo de integración para la simulación de 0.016 segundos, este escenario alcanza la estabilidad en menos de 500 pasos, por lo que se decidió simular hasta 1,000 pasos.

En la Tabla 6.10 se muestran los resultados de desempeño medidos en este experimento para cada uno de los 5 escenarios mencionados anteriormente, la implementación hace uso de un Octree con nivel de profundidad máximo 8. En estos resultados se observa que los procesos que más demandan recursos son UND y UNF, los cuales hacen uso del método de cálculo de vecindad propuesto en este trabajo.

En la Figura 6.20, se muestra el resultado de desempeño de los procesos UND y UNF para este experimento. Estos procesos muestran una tendencia lineal en su desempeño conforme se incrementan el número de partículas dentro de la simulación. Se observa que ambos procesos requieren una cantidad similar de tiempo, esto concuerda con

	P100000	P200000	P300000	P400000	P500000
UP [ms]	0.412	0.754	1.073	1.380	1.691
RNP [ms]	0.143	0.263	0.375	0.487	0.597
UOP [ms]	0.204	0.435	0.666	0.886	1.076
UND [ms]	2.720	10.412	23.986	43.331	61.739
UPR [ms]	0.022	0.037	0.053	0.072	0.101
UNF [ms]	2.175	8.231	19.306	35.723	52.039
UPEF [ms]	0.042	0.083	0.122	0.160	0.198
Loop time [ms]	6.252	20.769	46.181	82.662	118.081

Cuadro 6.10: Tabla con resultados de desempeño obtenido para los procesos de simulación de fluido en escenarios de nubes de puntos aleatorias. Se utilizó la implementación Opt y un Octree con nivel de profundidad máxima 8.

lo esperado, pues ambos procesos son similares y solo se diferencian en los valores calculados y número de operaciones requeridas para calcular los valores de densidad y fuerzas internas.

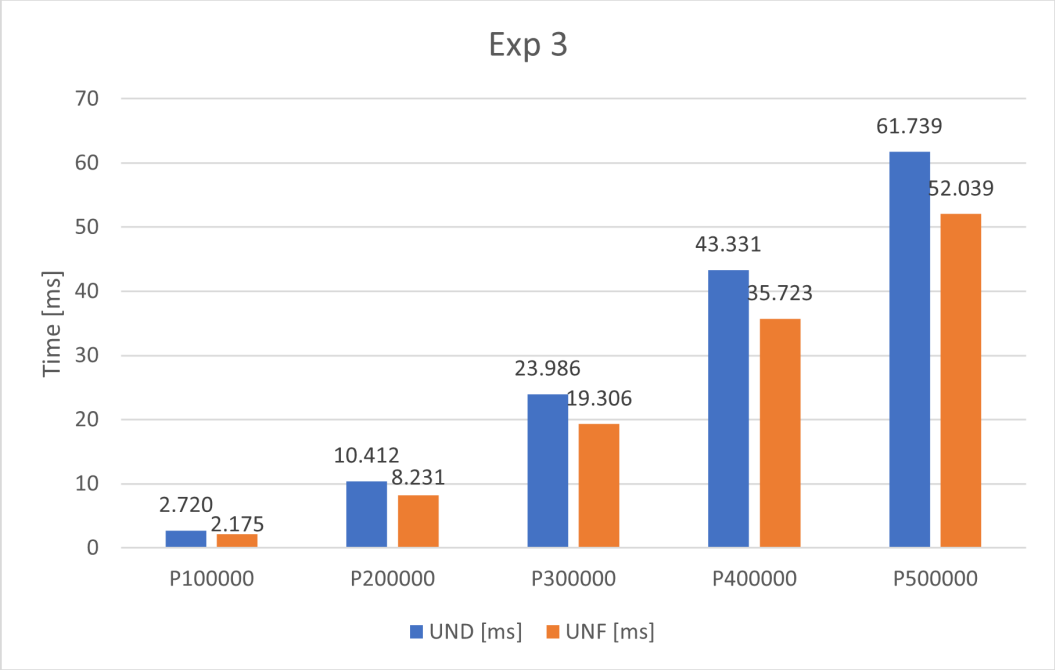

Figura 6.20: Gráfica de desempeño obtenido para los procesos UND y UNF, en escenarios de nubes de puntos aleatorias. Se utilizó la implementación Opt y un Octree con nivel de profundidad máxima 8.

6.4.2. Exp 5. Simulación de fluido con objeto estático

A el algoritmo presentado en la sección anterior se agregó un despliegue gráfico mediante interoperabilidad de CUDA y la biblioteca OpenGL. Como se mencionó anteriormente, el método de cálculo de vecinos implementado para este experimento es igual que el de Exp 4, del mismo modo se empleó el mismo equipo de cómputo y solo se agregó una etapa de despliegue gráfico con OpenGL.

Los escenarios empleados para este experimento fueron 3, estos escenarios constan de una plataforma rectangular sólida, la cual flota en el aire, es decir es un objeto estático; y un flujo de agua en forma de columna vertical, la cual se precipita hacia la plataforma por acción de la fuerza de gravedad. Los escenarios cuentan con diferente cantidad de partículas, ordenándolos de menor a mayor, el número de partículas con el que cuentan es: 21,288, 145,777 y 302,751. En la Figura 6.21 se muestran la secuencia de eventos de dos de los escenarios de simulados.

En la Tabla se muestran los resultados de desempeño de la simulación, se presentan los cuadros por segundo, FPS por sus siglas en inglés Frames per Second, tiempo de despliegue de cuadro, en inglés Frame Time y el total de partículas del escenario. Se observa que el método sigue teniendo un comportamiento lineal en su desempeño conforme se incrementa el número de partículas en la simulación. Hay que resaltar que el tiempo mostrado es promedio y existen periodos con despliegue gráfico más lento, pero también periodos donde el despliegue gráfico se acelera del mismo modo.

Escenario	1	2	3
FPS	333.750	66.041	28.823
Frame Time [ms]	2.996	15.142	34.695
Total Partículas	21,822	145,777	302,751

Cuadro 6.11: Tabla con resultados de desempeño obtenido para los tres escenarios de simulación de fluido con objeto estático.

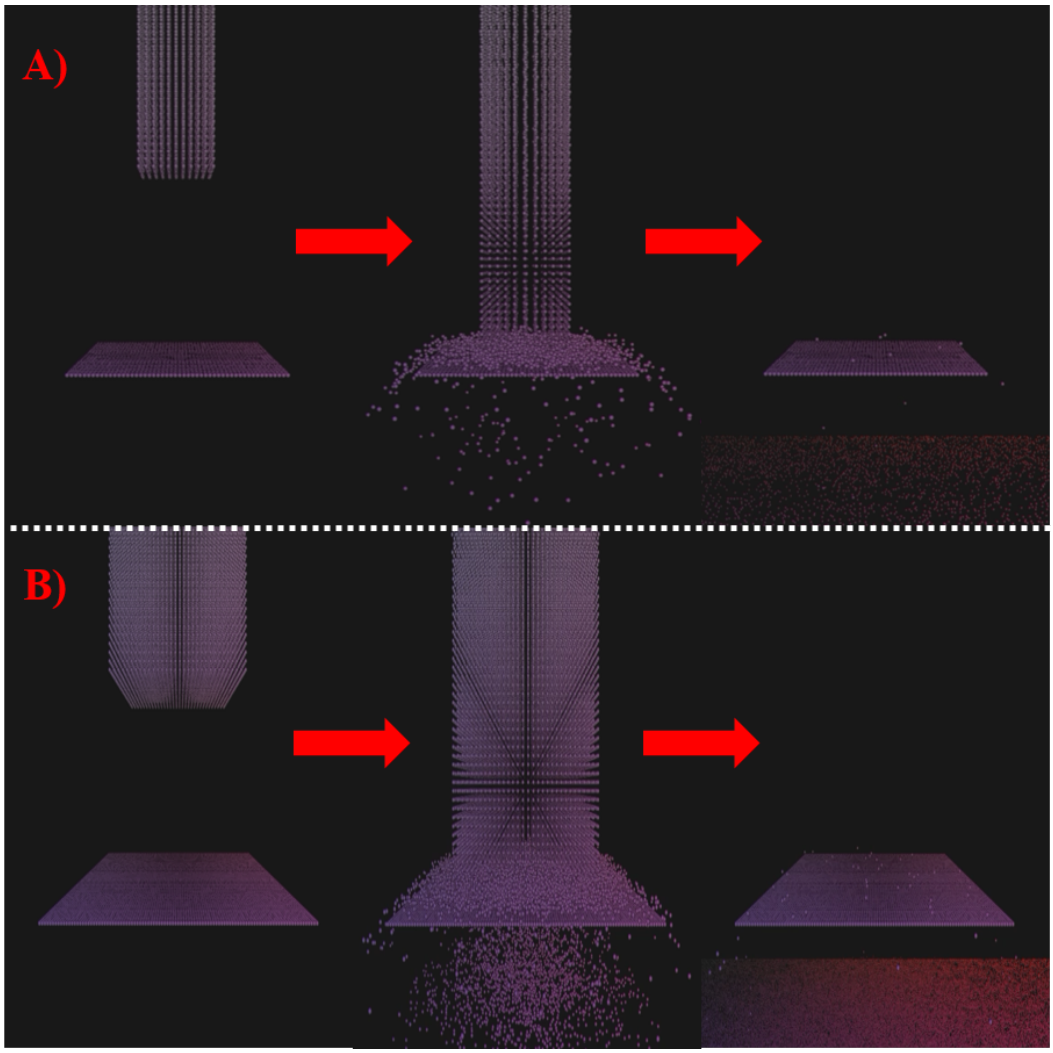

Figura 6.21: Secuencia de eventos de simulación de flujo con objeto estático. A) Escenario con 21,288 partículas. B) Escenario con 145,777 partículas.

6.4.3. Exp 5a. Simulación de fluido con objeto deformable (Viscoelástico)

En la siguiente fase de pruebas se agregó simulación de cuerpos deformables. La implementación de la simulación de cuerpos deformables emplea el método mencionado en el Capítulo 4, sin embargo esta implementación no es del todo beneficiada por el método propuesto para el cálculo de colisiones, por lo que la simulación no es completamente libre de mallas, es una implementación híbrida que mezcla SPH para simular la interacción de fluidos, cálculo de colisiones con el método propuesto en este trabajo y simulación de cuerpos visco elásticos mediante el método de masas y resortes.

El escenario empleado para está simulación es un modelo de plataforma como el del Exp 5, se retoma la misma geometría y consta de 21,288 partículas. No se midió desempeño para este método ya que es una implementación muy preliminar, así que se trata de una prueba de concepto. En la Figura 6.22 se muestra la secuencia de eventos para el escenario simulado.

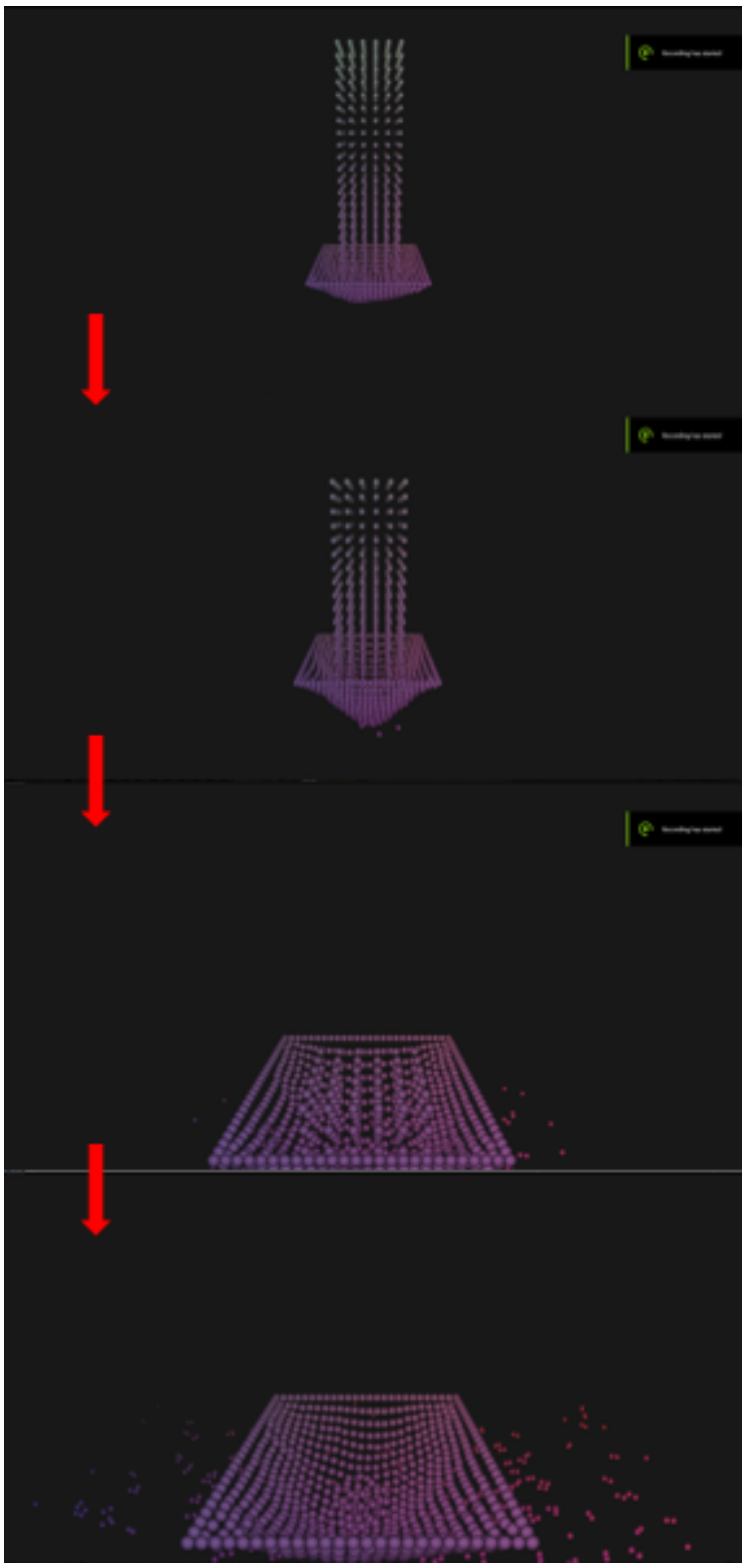

Figura 6.22: Secuencia de eventos de simulación de flujo con objeto deformable. Escenario con 21,288 partículas.

6.4.4. Exp 6. Simulación de fluidos (Sorted)

Para este experimento se desarrolló un algoritmo de simulación de fluidos y objetos estáticos, el cual utiliza la implementación de Sorted para el cálculo de vecindades.

El experimento consta de 3 escenarios, estos escenarios tienen las siguientes características:

A Plataforma rectangular con precipitación de columna de agua (302,751 partículas).

B Contenedor cilíndrico con precipitación de columna de agua (117,856 partículas).

C Embudo conectado a tubo en forma de S con precipitación de columna de agua (185,749 partículas).

El experimento fue ejecutado en un equipo de cómputo con las siguientes especificaciones:

- CPU: Intel i7-7700K.
- GPU: Nvidia GeForce GTX 1080Ti 11GB.
- Memoria: 32GB RAM DDR4.

En la Tabla 6.12 se muestran los resultados de desempeño medidos para la simulación usando un Octree con niveles de profundidad máxima 7, 8 y 9, es notable que, aunque se trata de un equipo con mayor capacidad de cómputo, la ejecución del escenario A de Exp 6 y el escenario 3 de Exp 5, los cuales son idénticos, muestra un mejor desempeño con la implementación SORTED. También se observa que entre más complejo sea el escenario, más se afecta el desempeño, la complejidad del escenario A, no se compara a los escenarios B y C. Adicional a esto se muestra en las Figuras 6.23, 6.24 y 6.25, las gráficas de desempeño en cada cuadro de simulación.

En la Figura 6.26 se muestra la secuencia de eventos para cada uno de los escenarios usados en este experimento, lo cual permite observar el aumento en la complejidad de la simulación.

	Level 7			Level 8			Level 9		
Scenario	AVG[ms]	MIN[ms]	MAX[ms]	AVG[ms]	MIN[ms]	MAX[ms]	AVG[ms]	MIN[ms]	MAX[ms]
A	81.114	16.770	275.445	16.118	9.530	47.714	10.758	9.663	12.178
B	75.670	18.811	279.155	8.355	4.802	19.508	4.127	3.553	5.444
C	138.866	35.991	428.803	12.593	7.656	30.978	5.942	5.479	7.749

Cuadro 6.12: Tabla con resultados de desempeño obtenido para simulación con implementación de cálculo de vecindad SORTED. Escenarios A, B Y C.

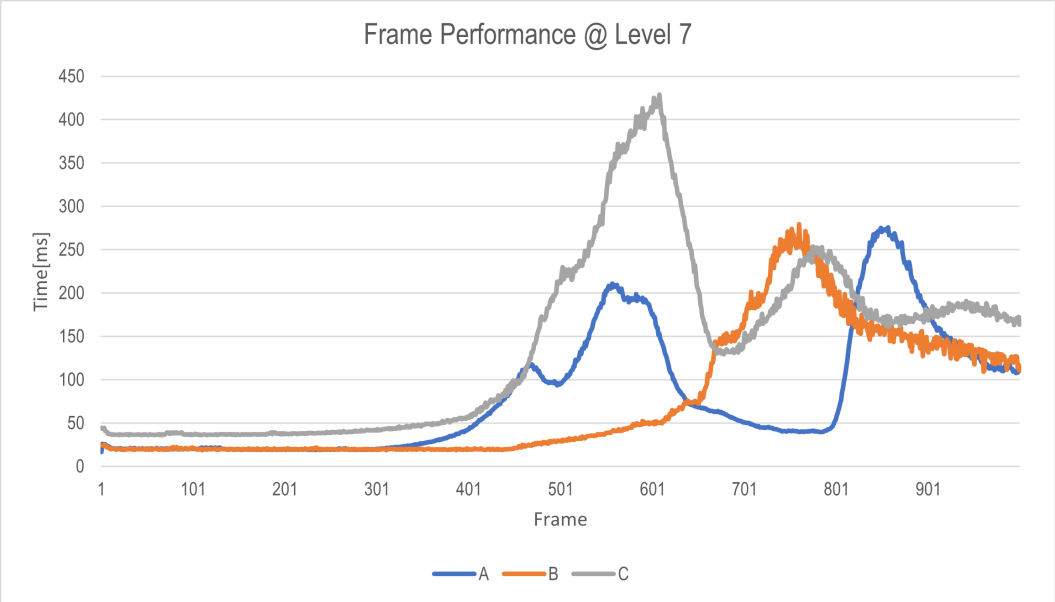

Figura 6.23: Desempeño por cuadro para cada uno de los escenarios usando un Octree con nivel de profundidad máxima 7.

Es necesario mencionar que la creación de los modelos para estos experimentos requirió de software adicional. Inicialmente se crearon modelos de mallas con el apoyo del software Blender [72], posteriormente dichos modelos fueron voxelizados mediante el software Binvox [73] y por último estos fueron transformados en partículas mediante un modulo agregado a el algoritmo desarrollado. En la Figura 6.27 se muestra un diagrama con el flujo descrito para la creación de los modelos, mientras que en la Figura 6.28 se muestran el resultado de transformar mallas a modelos de partículas.

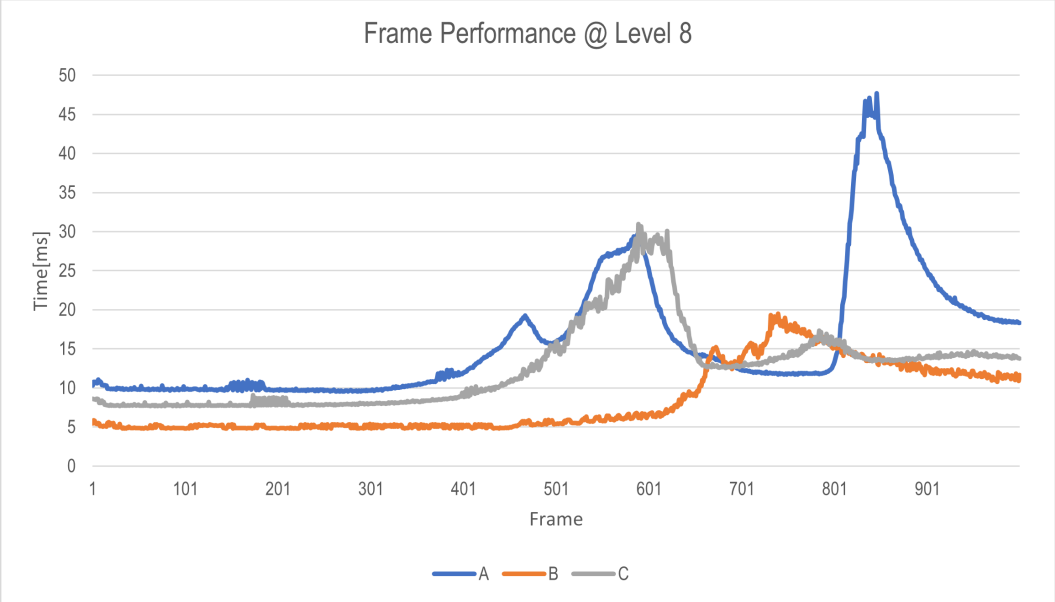

Figura 6.24: Desempeño por cuadro para cada uno de los escenarios usando un Octree con nivel de profundidad máxima 8.

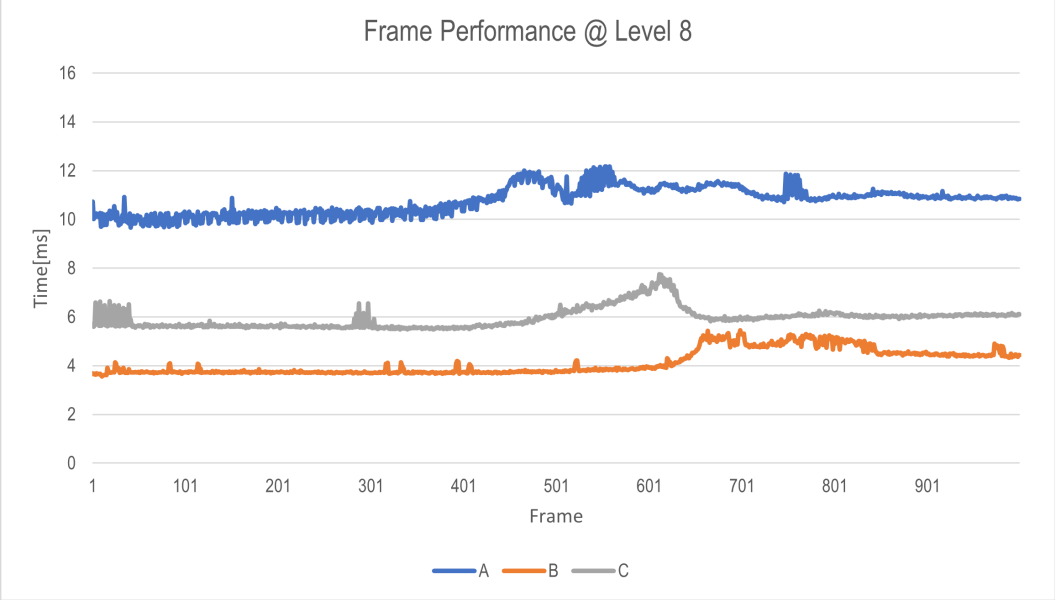

Figura 6.25: Desempeño por cuadro para cada uno de los escenarios usando un Octree con nivel de profundidad máxima 9.

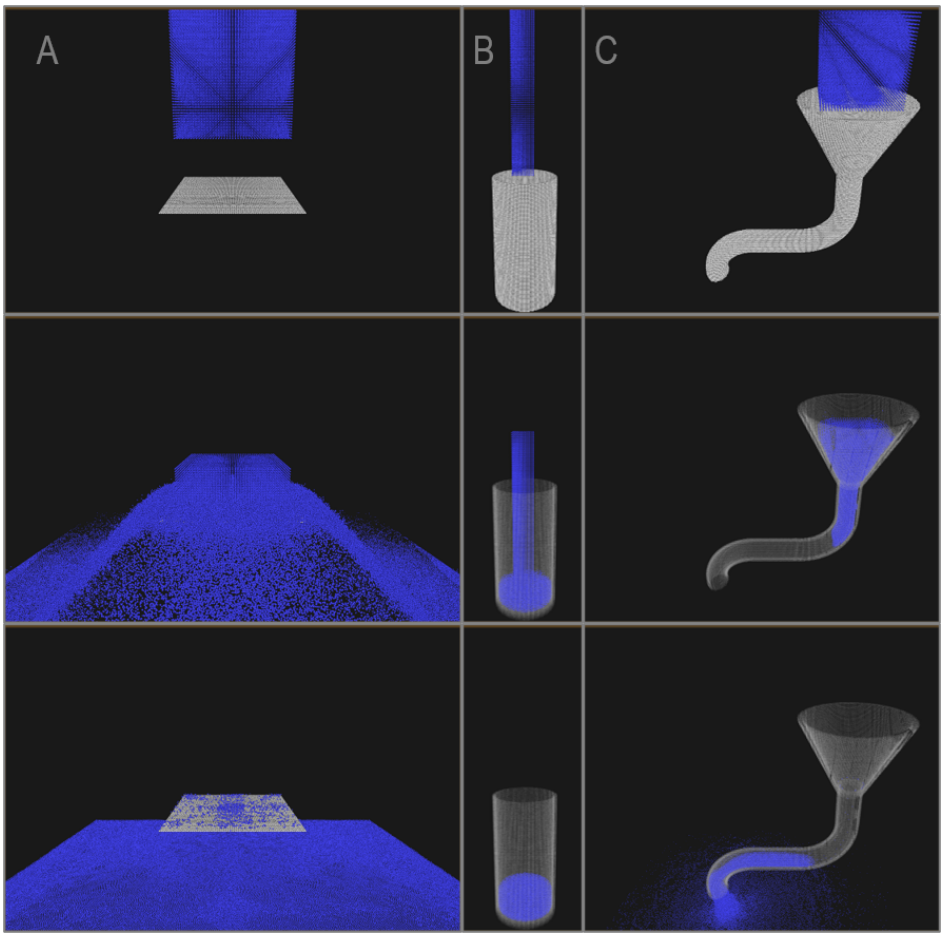

Figura 6.26: Secuencia de eventos de simulación con implementación de cálculo de vecindad SORTED. Escenarios A, B Y C.

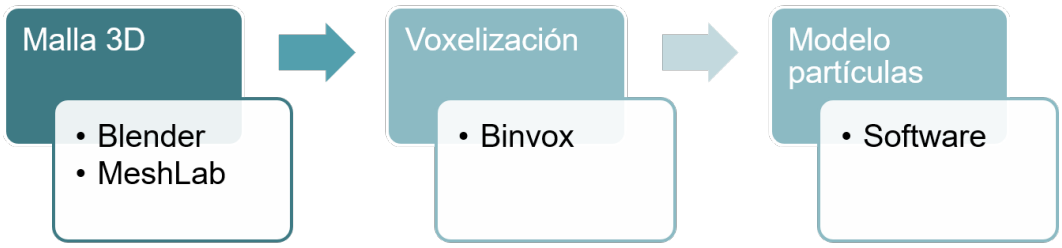

Figura 6.27: Diagrama de flujo para creación de modelos de partículas a partir de modelos de mallas triangulares.

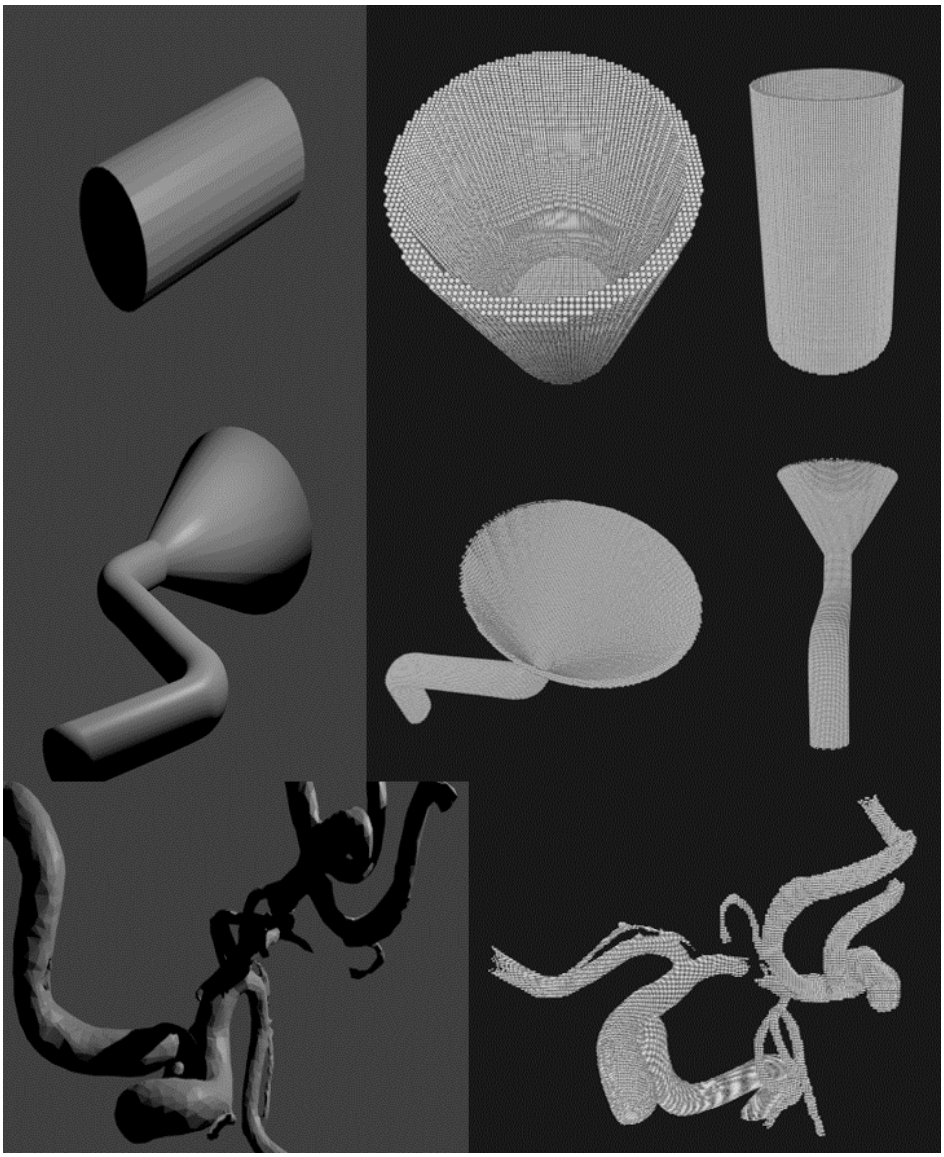

Figura 6.28: Ejemplos de modelos de mallas triangulares a modelos de partículas.

6.5. Exp 7. Estudio de forma con SPH.

En este experimento se desarrolló un software para el cálculo de forma usando SPH, este software emplea el método de cálculo de vecindad propuesto en este trabajo mediante la implementación Sorted, el cálculo de forma se realiza mediante el cálculo de valores de campo usando la aproximación de partículas suavizadas y los kernels de integración que se han mencionado en este trabajo. Para más información se puede consultar lo reportado por Soriano et al. [45].

El escenario usado para este experimento es un modelo que consta de 1,604,254 partículas. Este modelo fue proporcionado como una malla triangular de alta densidad, por lo que se empleó el proceso similar al descrito anteriormente para convertirlo en un modelo de partículas.

Los experimentos fueron ejecutados en dos equipos de cómputo con las siguientes especificaciones en hardware:

- Device A: Intel i7-7700HQ, Nvidia GeForce GTX 1070.
- Device B: Intel i7-7700K, Nvidia GeForce GTX 1080Ti.

Durante este experimento se analizó el desempeño de los procesos Smooth Particle Normal (SPN) y Smooth Particle Shape Descriptor (SPSD), los cuales hacen uso del método de calculo de vecindad con la implementación Sorted.

En la Tabla 6.13 se muestran los resultados de desempeño de los procesos SPN y SPSD, estos resultados se obtuvieron para cuatro Octrees con diferente nivel de profundidad máxima. En estos resultados se observó que conforme se incrementa el nivel de profundidad máxima del Octree, el desempeño mejora, esto se debe a que la cantidad de partículas contenidas en cada nodo se reduce. En la Tabla 6.14 se muestra el promedio de partículas por nodo dependiendo el nivel de profundidad máxima del Octree.

En la Figura 6.29 se muestran los resultados de cálculo de forma en modo gráfico y el modelo empleado para estos experimentos. Es evidente que conforme se incrementa el nivel de Octree se mejora la resolución del método, pero para poder hacer un cálculo de forma en los detalles subir el nivel del Octree es indispensable.

Proceso	Device A	Device B	Nivel Octree
SPN [ms]	488.99	320.03	6
SPSD [ms]	523.18	344.97	
SPN [ms]	2573.87	1826.55	7
SPSD [ms]	2546.46	1863.71	
SPN [ms]	488.99	320.03	8
SPSD [ms]	523.18	344.97	
SPN [ms]	94.37	62.68	9
SPSD [ms]	107.07	72.04	

Cuadro 6.13: Tabla con resultados de desempeño obtenido para estudio de forma con implementación de cálculo de vecindad SORTED.

Nivel Octree	Promedio Partículas
6	227.74
7	58.44
8	15.39
9	4.35

Cuadro 6.14: Numero promedio de partículas en nodos de Octree a diferentes niveles de profundidad máxima.

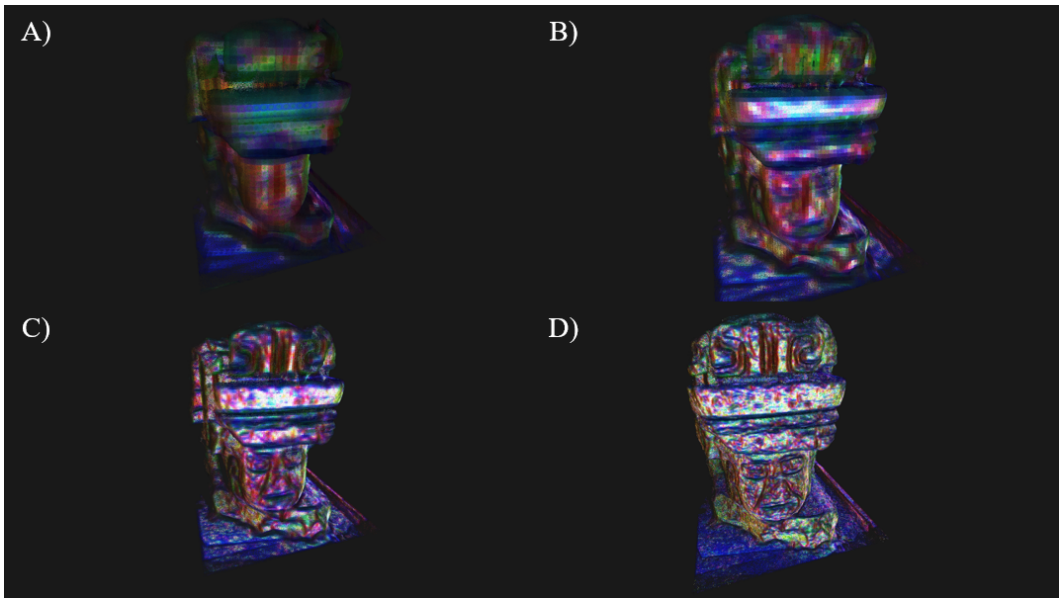

Figura 6.29: Resultado gráfico de experimento de estudio de forma con SPH. Modelo con 1,604,254 partículas. A) Nivel 6. B) Nivel 7

6.6. Exp 8. Simulación de flujo en formas diversas (poros).

Para evaluar el método con respecto a la forma de los objetos que conducen el flujo se realizó una prueba para simular flujo de agua saturada en diversos poros con diversas formas (tubulares, verticales, horizontales y amorfos).

Los poros se obtuvieron de un núcleo de suelo intacto de 8.5 cm de diámetro y 15.0 cm de largo, el cual fue escaneado utilizando un tomógrafo (TomRX Solutions) disponibles en el laboratorio 3SR en Grenoble. La distancia del detector de origen se estableció en 752 mm, la distancia de origen a objeto en 347 mm para una resolución de voxel de 0.087 mm en (x, y, z).

El volumen de imágenes CT obtenidas I define un volumen $V \in \mathbb{Z}$. La colección de todos los voxeles $\Gamma \in V$ con tamaño igual a $[1515, 1515, 1699]$ voxeles en las direcciones (x, y, z), respectivamente. El volumen fue binarizado y los poros etiquetados utilizando la aplicación SMAS (http://www.biocomlab.com/apps/SMAS.html). Del volumen segmentado se seleccionaron 91 poros debido a que eran los más grandes y permitían definir un punto para el flujo de entrada y uno para el de salida, estos poros representan los macroporos del núcleo de suelo. En la Figura 6.30 se muestra el resultado de la segmentación de los poros.

De estos 91 poros, 48 poros pueden ser clasificados como tubulares o compuestos por múltiples estructuras tubulares y 43 poros como amorfos.

La configuración para la simulación numérica considera un radio de interacción r_i de 217 μm, la masa m_i de la partícula es equivalente a la de una esfera de radio igual a la mitad de la resolución del voxel $(3,45 \times 10^{-7} g)$. La viscosidad dinámica del agua se estableció en $8,90 \times 10^{-4} Pa \cdot s$. Las fuerzas externas son la gravedad y $8,70 \times 10^{-6} Pa$ de diferencial de presión para las condiciones de los límites se impone una condición antideslizante.

El flujo volumétrico es calculado para cada poro cuando se cumple la condición de que están saturados, y se prueba si el flujo volumétrico en cada sección transversal $(\overline{V_f(\Phi_p)})$ es similar y la simulación es estable. Para obtener las secciones transversales se utilizan los esqueletos de cada estructura, la Figura 6.31 muestra algunos poros con

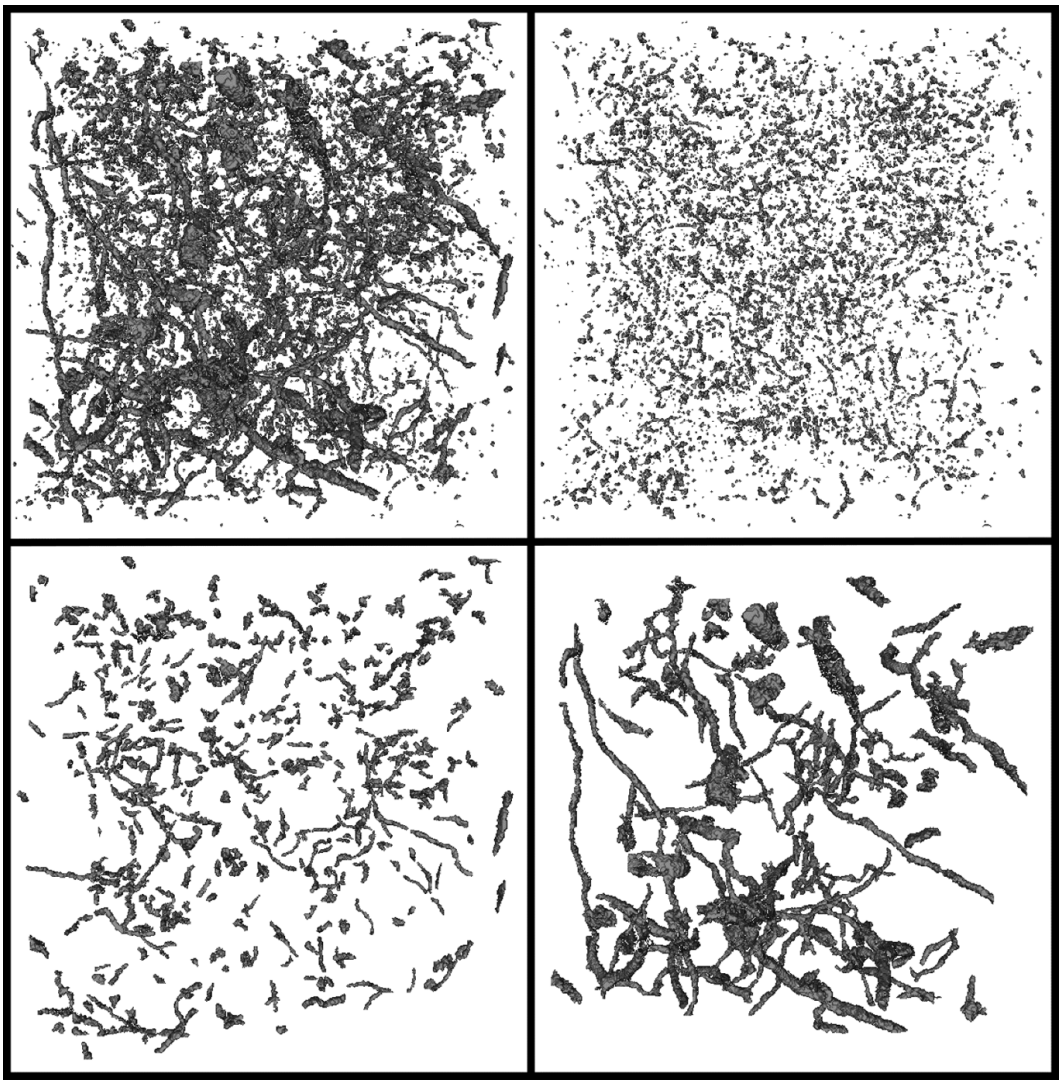

Figura 6.30: Estructuras de poros que forman el volumen. Arriba a la izquierda: Todos los poros 3D dentro del volumen. Arriba a la derecha: dimensión de los poros del tamaño 0 a 100 voxeles. Abajo a la izquierda: Tamaño de los poros de 100 a 1000 voxeles. Abajo a la derecha: Tamaño del poro de 1000 voxeles al tamaño máximo, este grupo esta formado por los 91 poros utilizados para evaluar la implementación de SPH.

su esqueleto.

En la Figura 6.32 se muestra el resultado de simulación utilizando el valor de flujo volumétrico.

La desviación estándar del flujo volumétrico se calcula en cada sección transversal para probar la estabilidad del flujo, la Figura 6.33 muestra el flujo volumétrico medio

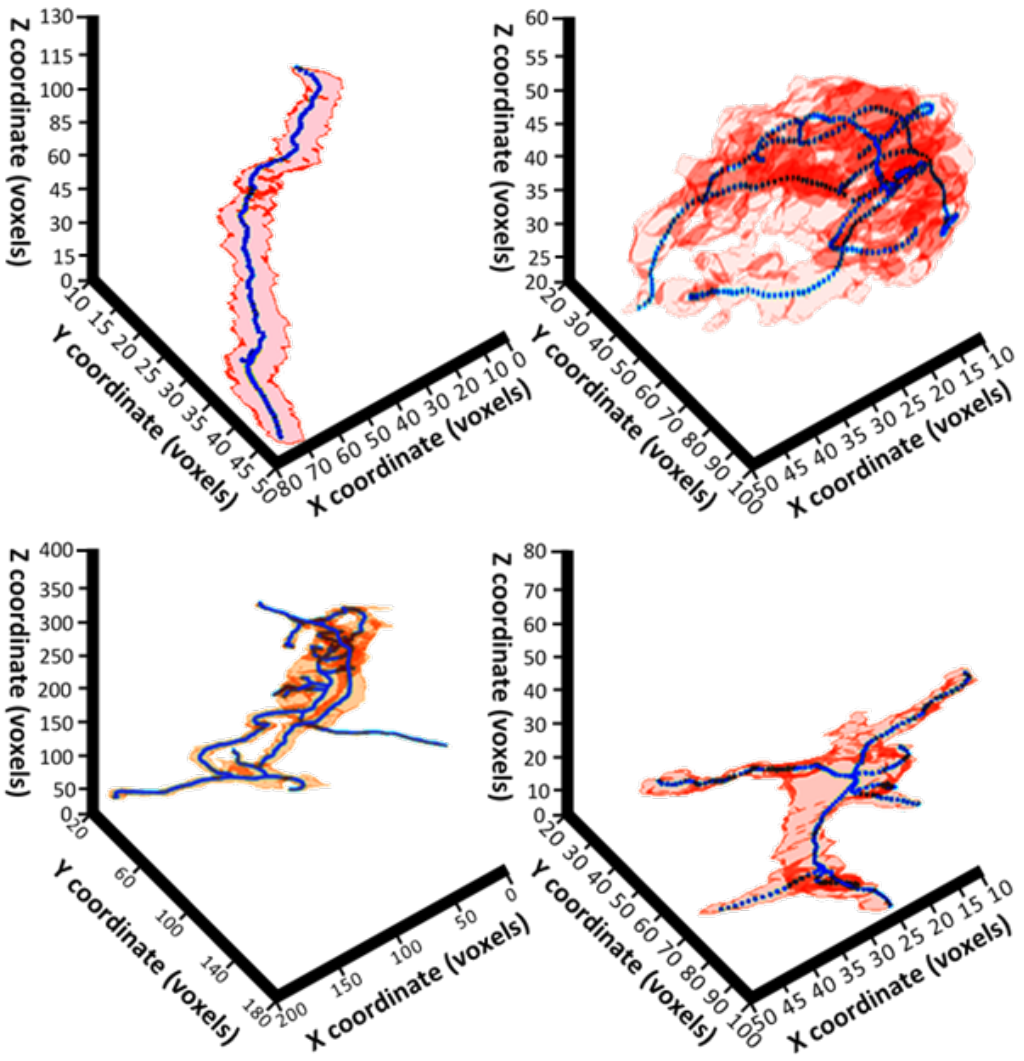

Figura 6.31: Volúmenes de poros con sus estructuras de esqueleto internas, el esqueleto se utiliza para definir las áreas de sección transversal con las que se calcula el flujo volumétrico.

y la desviación estándar asociada de cada poro.

Se puede observar que las condiciones de flujo saturado diversos poros tienen diferente eficiencia de conducción y por lo tanto capacidad de flujo volumétrico, pero en todos se logra mantener un flujo estable cuando se definen las condiciones de entrada para mantener el poro saturado.

Utilizando el software SMAS nos es posible también presentar los resultados en forma de eficiencia de conducción de flujo. Para definir la medida de eficiencia de la

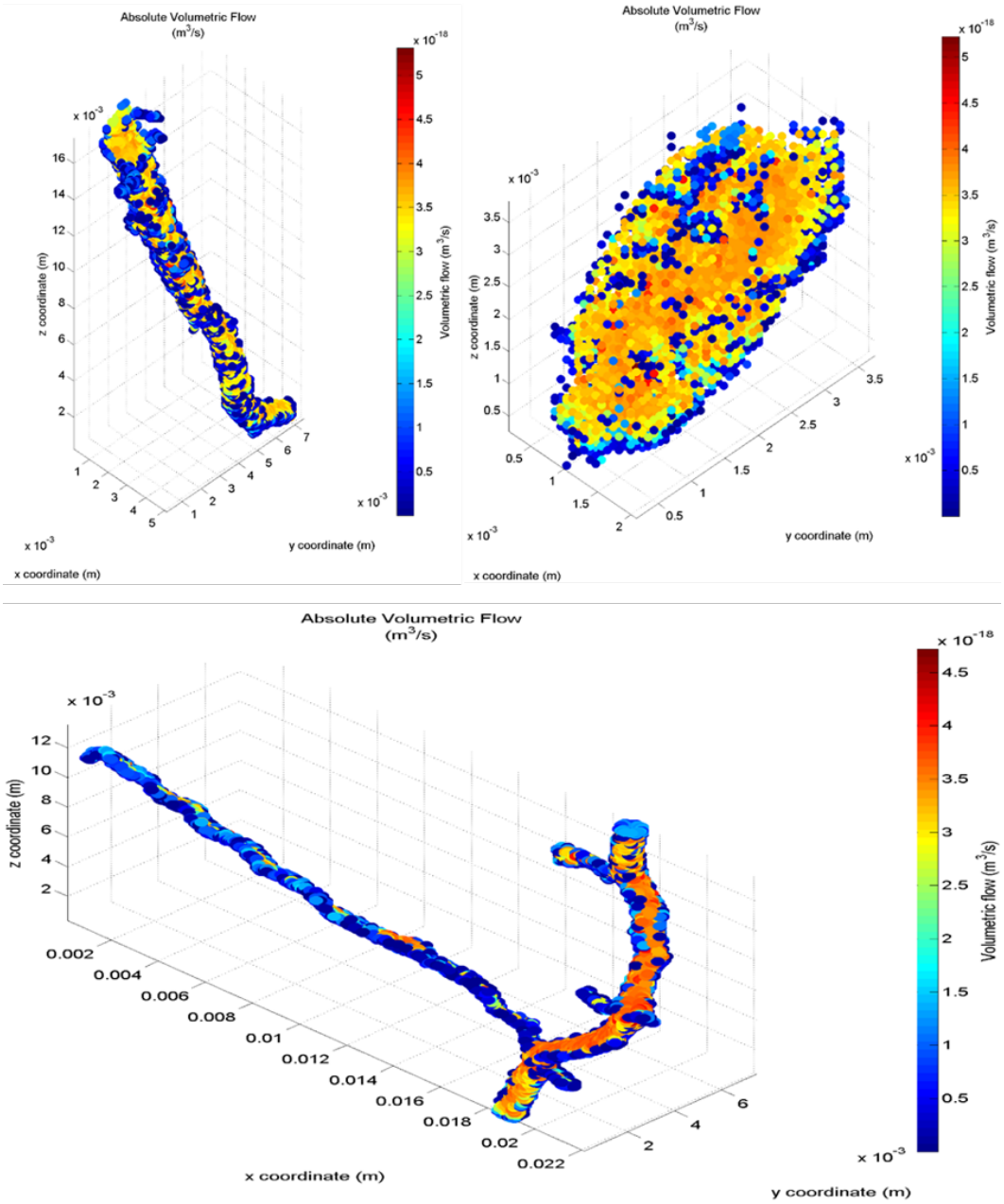

Figura 6.32: Resultado de simulación numérica para diferentes tipos de estructura de poros. La figura muestra un poro (arriba a la izquierda) con una estructura más similar a una estructura tubular, (arriba a la derecha) muestra una estructura de poro con una forma similar a un esferoide donde las condiciones de entrada y salida son más difíciles de definir. (Abajo) Estructura de poros con dos estructuras tubulares que se cruzan.

conducción de flujo en cada poro un cilindro circular equivalente $Q_{cyl}(\Phi_p)$ definido utilizando el radio promedio de las secciones transversales de cada poro $R_{cyl}(\Phi_p)$. El flujo volumétrico del cilindro equivalente se obtiene de forma analítica utilizando la

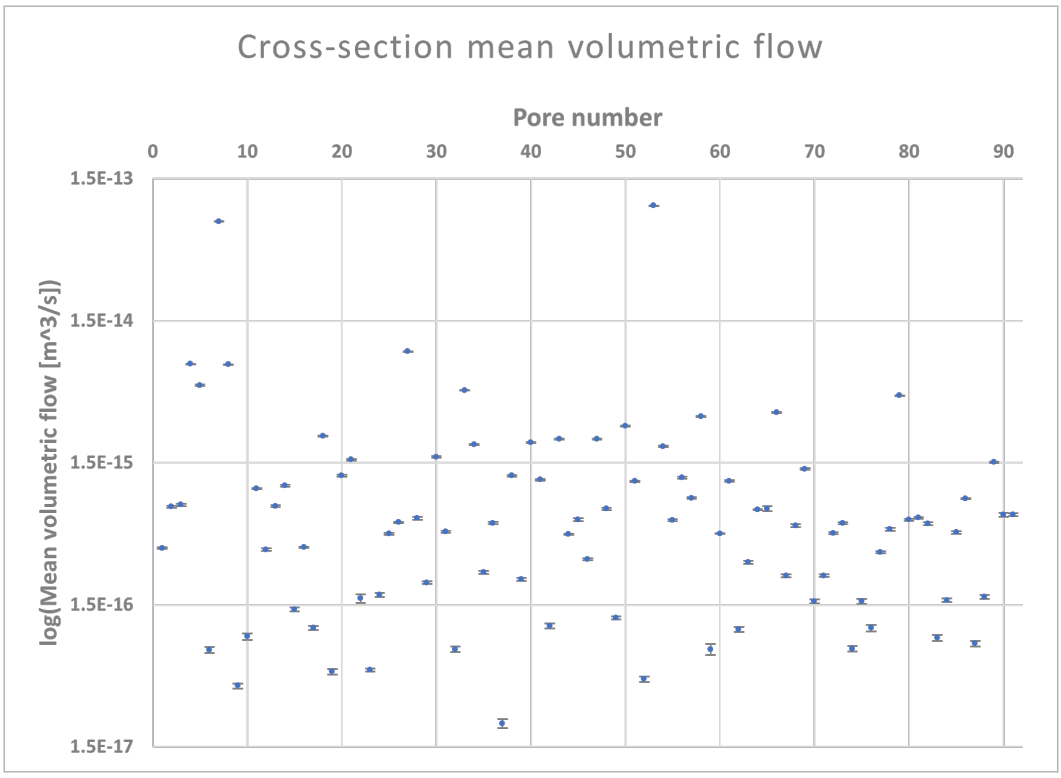

Figura 6.33: Volúmenes de poros con sus estructuras de esqueleto internas, el esqueleto se utiliza para definir las áreas de sección transversal con las que se calcula el flujo volumétrico.

ecuación de Manning (6.2) [74].

$$Q_{cyl}(\Phi_p) = (\frac{1}{M_n})(\frac{\pi(2R_{cyl}(\Phi_p))^2}{4})(\frac{R_{cyl}(\Phi_p)}{2})^{\frac{2}{3}}, \tag{6.2}$$

donde $M_n$es el coeficiente de rugosidad de Manning, se utiliza el valor de 0,022 para definir un canal de tierra limpia.

Estos cilindros equivalentes se consideran la forma más eficiente para conducir el flujo. Así, la relación del flujo volumétrico promedio ($\overline{V_f(\Phi_p)}$) respecto del flujo volumétrico del cilindro circular equivalente $Q_{cyl}(\Phi_p)$ define la medida de la eficiencia de cada poro, esto se muestra en la ecuación (6.3).

$$E_{cyl}(\Phi_p) = \frac{\overline{V_f(\Phi_p)})}{Q_{cyl}(\Phi_p)} \tag{6.3}$$

En la Figura 6.34 se muestran los resultados obtenidos para lo poros analizados, donde el poro con al forma más tubular fue el más eficiente.

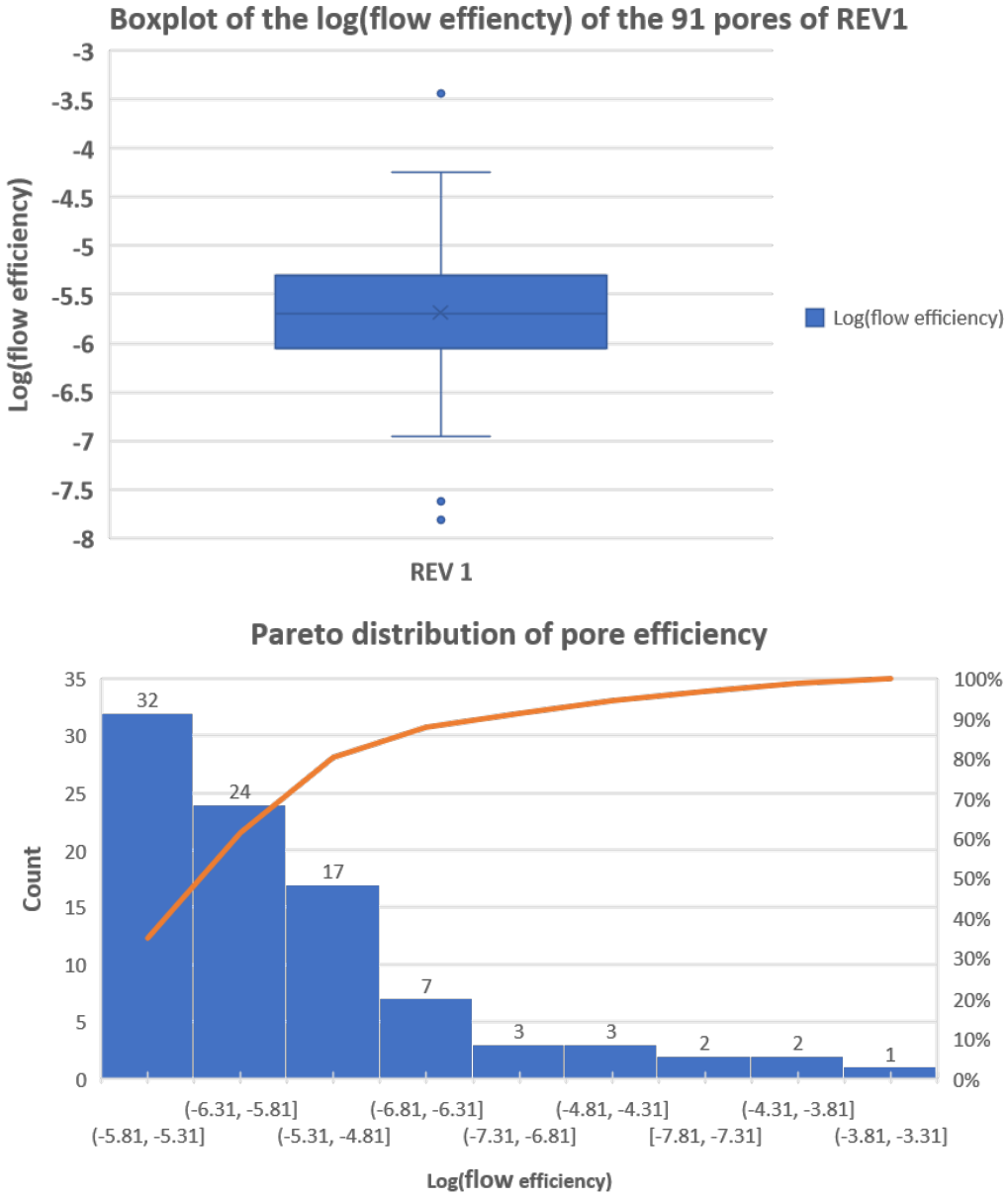

Figura 6.34: (Arriba) Distribución de boxplot de la eficiencia de flujo de poros en escala logarítmica, el poro 7 tiene la mayor eficiencia (su forma es la mas tubular) y los poros 18 y 90 la menor (ambos amorfos). (Abajo) Distribución de Pareto, sólo 9 poros están por encima del 10 % de la eficiencia.

Capítulo 7

Conclusiones

En este trabajo se ha realizado investigación en simulación libre de mallas por partículas suavizadas (SPH), este método, al igual que todos los métodos basados en partículas, requieren realizar el cálculo de vecindades en cada una de las partículas que representa el dominio de estudio, para así poder aplicar el modelo matemático que permite realizar la simulación de un fenómeno especifico. Respecto al menester del cálculo de vecindad, es necesario dejar claro el motivo para el cual se hace investigación y proponen enfoques nuevos para dicho método, el enfoque puede variar desde evitar problemas de diversa índole en el método, hasta mejorar el desempeño, resolución o exactitud de este, pasando por la inclusión de nuevas capacidades o expansión de los modelos matemáticos.

En el caso de este trabajo el problema abordado es el método de cálculo de vecindad con el objetivo de mejorar el desempeño al emplear arquitecturas de cómputo en paralelo, como lo es CUDA. Este problema se ha resuelto con éxito mediante enfoques que dividen el espacio y ordenan el conjunto de elementos que comprenden el dominio del mismo; es necesario mencionar que el cálculo de vecindad es una variación del problema del vecino más cercano y este se relaciona al problema del vendedor viajero (TSP). Dado lo anterior es que se propuso un método que incluye división y ordenamiento espaciales, pero la aportación más importante es agregar indexado a la estructura que nos permite dividir y ordenar, este indexado se realizó mediante el uso de claves Morton.

Las estructuras de datos empleadas de manera predominante para resolver problemas donde se divide el espacio son los arboles, en el caso especifico del espacio tridimensional, se emplea la estructura de datos Octree. Esta misma estructura de datos es la que se usa en este trabajo.

Al agregar indexado mediante claves Morton es que se tiene la posibilidad de recorrer el Octree no solo mediante la relación jerárquica de padre e hijo (guardada en punteros en memoria), si no también con la codificación de las claves, las cuales permiten calcular las claves de todos los nodos adyacentes a un nodo especifico.

El método propuesto en este trabajo para el cálculo de las claves Morton de los nodos vecinos requiere únicamente de un numero acotado de operaciones binarias, por lo que este método es ideal para arquitecturas que tengan gran capacidad de cómputo aritmético lógico, como lo son las arquitecturas en paralelo que usan cómputo en Unidad de procesamiento gráfica de propósito general, como lo es CUDA. Como resultado tenemos que el método aprovecha de manera eficiente el poder de cómputo de las GPUs mediante un algoritmo de complejidad constante $O(c)$. Esto es ideal ya que en este trabajo se demostró que la mayor ventaja de las capacidades de cómputo de las GPU se encuentra en el cálculo de intensivo.

Esto último es sumamente importante, ya que, si el Octree propuesto en este trabajo fuera convencional, dependería mayormente de apuntadores y memoria, lo cual se demostró es menos eficiente para implementaciones en CUDA, al menos para las generaciones actuales de tarjetas gráficas; aunque ese aspecto mejore a futuro, se debe considerar que también es posible que mejoren las capacidades de cómputo intensivo, por lo que el método propuesto seguirá sacando provecho de dicha arquitectura.

Las claves Morton pueden indexar cualquier número de dimensiones, así como las estructuras de datos de árbol, pueden dividir el espacio n dimensional que se requiera. Dado lo anterior se puede afirmar que el método propuesto se puede aplicar en cualquier número de dimensiones, adicional a eso, el método de cálculo de vecindad no incrementa su complejidad conforme se incrementan las dimensiones, claro que al modificarse la adyacencia conforme se modifica el numero de dimensiones, se tendrá un impacto. Como se mencionó anteriormente, esto es posible gracias a las propiedades de la estructura de datos de árbol y la geometría de curva Z que se obtiene mediante el indexado por clave Morton.

El método propuesto se validó contra modelos analíticos empleando la implementación descrita de SPH con ecuaciones de Navier-Stokes. Como resultado se obtuvo que la solución interpolada mediante SPH cumple con lo esperado por la solución analítica, esto era de suma importancia, ya que, el método acota la región de trabajo para los cálculos de las partículas, aunque esto se puede mitigar con variaciones en profundidad del Octree, es deseable que esto no sea necesario.

Este método hace posible acelerar el cálculo de propiedades de campo para partículas al emplear las capacidades de cómputo de las arquitecturas en paralelo que proporcionan las tarjetas gráficas actuales. La eficiencia que se consigue permite tener simulaciones en tiempo real al incrementar el nivel de profundidad del Octree, pero también se pueden hacer cálculos más elaborados, al reducir el nivel de profundidad del Octree. Esta versatilidad hace que el método propuesto tenga múltiples aplicaciones, como la simulación y el cálculo de forma, las cuales se mostraron en este trabajo.

7.1. Discusión

Es necesario considerar la naturaleza lagrangiana de las simulaciones libres de malla tiene algunos problemas cuando se quiere estudiar un sistema completo.

Algunas de las limitaciones de la aproximación lagrangiana tiene que ver con las condiciones de frontera y el simular un flujo dentro de un sistema complejo. En los métodos eulerianos, como lo es FEM en simulación de fluidos, se puede definir un dominio y dividirlo en diversos elementos con diversas condiciones para simular un fenómeno de flujo complejo, si quisiéramos hacer lo mismo en un método lagrangiano tendríamos que definir a un dominio con partículas que de la misma forma representen los diferentes materiales utilizados y esto aumentaría el numero de partículas que definen las condiciones de frontera al fenómeno que se esta simulando.

Respecto al método presentado se presentaría un impacto al desempeño, debido a que el Octree sería muy denso, donde la mayoría de los nodos estarían llenos; esto se observo en los experimentos, donde entre más disperso se encuentran los nodos del Octree ocupados por partículas, mejor es el desempeño del método propuesto, mientras que una saturación de todos los nodos da como resultado una caída en el desempeño.

Para resolver algunos problemas de este estilo se utilizan condiciones de frontera con partículas denominadas fantasmas que ayudan a definir las condiciones de frontera de forma local, estas se declaran usando las fronteras de los conductos para los flujos y ayudan a introducir diversas condiciones en diversos puntos de la simulación.

Por otro lado, aunque se lograse construir una simulación de flujo lagrangiana que no requiriera llenar todo el espacio, la cual pueda simular de manera idéntica el mismo fenómeno que una simulación de flujo euleriana, la resolución de la simulación lagrangiana depende del número de partículas, mientras que la euleriana puede interpolar y determinar con mayor precisión y exactitud un punto especifico y de interés en el espacio. Esto pone en desventaja a la simulación lagrangiana. Aunque la simulación lagrangiana permite observar de manera más natural el evento simulado, en la práctica el estudio se realiza sobre puntos específicos, lo cual es una ventaja para la simulación euleriana.

Además de esto, si se desea analizar la información producida por una simulación de flujo lagrangiana se cae en problemas de grandes cantidades de datos (*Big Data*), por lo que es necesario agregar un método que permita acotar la región de interés para la generación de datos, aunque también es posible usar el Octree para condensar la información resultante de la simulación de flujo.

Aunque el método propuesto logra aprovechar de manera eficiente los recursos de cómputo de las arquitecturas en paralelo, esto no permite que las simulaciones de flujo del tipo lagrangiano puedan sustituir en todo escenario a las simulaciones de flujo eulerianas. Es necesario reconocer que la propuesta de mejora en desempeño no es determinante, debido a que en todo proyecto donde se recurre al uso de simulaciones, la elección del método recae en el modelo y resolución que se emplea para recrear la realidad, lo cual está directamente relacionado con lo que se desea estudiar del fenómeno que es simulado.

7.2. Trabajo futuro

Durante la realización de este trabajo, se empezó a observar que el indexado por calve Morton y la estructura de datos Octree, comparten propiedades, como lo es el

ordenamiento espacial. Esta peculiaridad sugiere que debe ser posible que la estructura de datos Octree pueda ser implementada únicamente mediante el uso de secuencias de claves Morton. Sin embargo, esa investigación está pendiente debido a limitaciones de tiempo. Además, este problema no es trivial y requiere ingenio, pues no es obvio como es que la clave Morton puede contener la información de jerarquización e indexado de un Octree.

En cuanto al aspecto de simulación de cuerpos deformables, en este trabajo no se pudo conseguir una simulación que fuera completamente libre de mallas, aunque la eficiencia del método de cálculo de vecindad proporciona la capacidad de agregar mayor complejidad a los cálculos, se requiere un experto en modelado de dicho comportamiento, para tener una simulación adecuada.

El cálculo de colisiones es muy rudimentario, por lo que se requiere refinar el método, pero esto puede resolverse una vez se adopten métodos que simulen sin emplear mallas y la representación por superficies no dependa de mallas, sino sea directamente por conjuntos de partículas.

Apéndice A

Repositorios de código

Por motivos de seguridad, los repositorios pueden cambiar a modo privado sin previo aviso, por lo que se puede solicitar acceso mediante un correo electrónico enviado a la dirección:

d.soriano.valdez@gmail.com

Código para calculo de vecindades usando método de Octree:

https://gitlab.com/d.soriano.valdez/ParticleOctree

Simulación de flujo con SPH v1.0:

https://gitlab.com/d.soriano.valdez/SPHOctree

Simulación de flujo con SPH v.2.0:

https://gitlab.com/d.soriano.valdez/sphoctreev2

Simulación de flujo con SPH v2.1:

https://github.com/davidsorval/SPH_Normalized

Apéndice B

Método de búsqueda de vecinos

```
1  //Funcion para calcular la clave Morton de un nodo adyacente por cara
2  //En caso de existir el vecino, flag = true y nmk es su clave morton.
3  //maskIni 0x1 X(R,L) 0x2 Y(U,D) 0x4 Z(F,B)
4  //AndRes 0x0 X+(R)Y+(U)Z+(F) 0x1 X-(L) 0x2 Y+(D) 0x4 Z-(B)
5  __device__ void getNodeNeighbor(unsigned int mk, unsigned int maskIni,
       unsigned int andRes, unsigned int depth, unsigned int& nmk, bool&
       flag)
6  {
7    unsigned int infLimit = 10 - depth;
8    unsigned int mask = 0;
9    unsigned int myRes = 0;
10   unsigned int test = mk << 2;
11   test = test >> 2; flag = false; nmk = mk;
12   for (int j = infLimit; j < 10; j++)
13   {
14     mask = (maskIni) << (j * 3);
15     myRes = (andRes) << (j * 3);
16     if ((test & mask) == myRes)
17     {
18       j = 10;
19       flag = true;
20     }
21     nmk = nmk ^ mask;
22   }
23 }
```

Listing B.1: Método de búsqueda de vecinos por clave Morton (CUDA).

```
//Funcion para calcular la clave Morton de un nodo adyacente por cara
//En caso de existir el vecino, flag = true y nmk es su clave morton.
//maskIni 0x1 X(R,L) 0x2 Y(U,D) 0x4 Z(F,B)
//AndRes 0x0 X+(R)Y+(U)Z+(F) 0x1 X-(L) 0x2 Y+(D) 0x4 Z-(B)
void OctreeHelper::getNodeNeighbor(unsigned int mk, unsigned int
    maskIni, unsigned int andRes, unsigned int depth, unsigned int&
    nmk, bool& flag)
{
  unsigned int infLimit = 10 - depth;
  unsigned int mask = 0;
  unsigned int myRes = 0;
  unsigned int test = mk << 2;
  test = test >> 2; flag = false; nmk = mk;
  for (int j = infLimit; j < 10; j++)
  {
    mask = (maskIni) << (j * 3);
    myRes = (andRes) << (j * 3);
    if ((test & mask) == myRes)
    {
      j = 10;
      flag = true;
    }
    nmk = nmk ^ mask;
  }
}
```

Listing B.2: Método de búsqueda de vecinos por clave Morton (C++).

Apéndice C

Estructuras de datos para experimento 1

```
struct OctreeNodo
{
  unsigned int morton_key;
  unsigned int particleList[NODE_PARTICLE_SIZE];
  unsigned int p_count;
  float c_x;
  float c_y;
  float c_z;
  char neighbor_amt;
  unsigned int neighbors[NODE_NEIGHBOR_SIZE];
};

struct Particle
{
  float x;
  float y;
  float z;
  unsigned int m_x;
  unsigned int m_y;
  unsigned int m_z;
  float c_x;
  float c_y;
  float c_z;
  unsigned int morton_key;
```

```
};
```

Listing C.1: Implementación LUT (Nodos y partículas).

```
struct OctreeNodoAlt
{
  unsigned int morton_key;
  unsigned int particleList[NODE_PARTICLE_SIZE];
  unsigned int p_count;
};

struct ParticleAlt
{
  float x;
  float y;
  float z;
  unsigned int m_x;
  unsigned int m_y;
  unsigned int m_z;
  float c_x;
  float c_y;
  float c_z;
  unsigned int morton_key;
};
```

Listing C.2: Implementación Alternative (Nodo y partículas).

```
struct OctreeNodoOpt
{
  unsigned int morton_key;
  unsigned int particleList[NODE_PARTICLE_SIZE];
  unsigned int p_count;
  unsigned int statusR;
  unsigned int statusN;
};

struct ParticleOpt
{
  float x;
  float y;
  float z;
  unsigned int m_x;
  unsigned int m_y;
```

```
  unsigned int m_z;
  float c_x;
  float c_y;
  float c_z;
  unsigned int morton_key;
  unsigned int morton_key_old;
};
```

Listing C.3: Implementación Opt (Nodo y partículas).

```
struct OctreeNodoRec
{
  unsigned int morton_key;
  unsigned int particleList[NODE_PARTICLE_SIZE];
  unsigned int p_count;
  unsigned int statusR;
  unsigned int statusN;
};

struct ParticleRec
{
  float x;
  float y;
  float z;
  unsigned int m_x;
  unsigned int m_y;
  unsigned int m_z;
  float c_x;
  float c_y;
  float c_z;
  unsigned int morton_key;
};
```

Listing C.4: Implementación Rec (Nodo y partículas).

Apéndice D

Funciones para uso de claves Morton (CUDA)

```
__device__ unsigned int p1b2(unsigned int n)
{
  n = (n ^ (n << 16)) & 0xff0000ff; //(1)
   11111111000000000000000011111111
  n = (n ^ (n << 8)) & 0x0300f00f; //(1)
   11000000001111000000001111
  n = (n ^ (n << 4)) & 0x030c30c3; //(1)
   11000011000011000011000011
  n = (n ^ (n << 2)) & 0x09249249; //(1)
   10010010010010010010010010010
  return n;
}

__device__ unsigned int c1b2(unsigned int x)
{
  x &= 0x09249249;                  // x = ---- 9--8 --7- -6-- 5--4
   --3- -2-- 1--0
  x = (x ^ (x >> 2)) & 0x030c30c3; // x = ---- --98 ---- 76-- --54
   ---- 32-- --10
  x = (x ^ (x >> 4)) & 0x0300f00f; // x = ---- --98 ---- ---- 7654
   ---- ---- 3210
  x = (x ^ (x >> 8)) & 0xff0000ff; // x = ---- --98 ---- ---- ----
   ---- 7654 3210
  x = (x ^ (x >> 16)) & 0x000003ff; // x = ---- ---- ---- ---- ----
```

```
    --98 7654 3210
  return x;
}

__device__ unsigned int MK3(unsigned int x, unsigned int y, unsigned
    int z, unsigned int depth)
{
  unsigned int mk = (1 << 30) + (p1b2(z) << 2) + (p1b2(y) << 1) + (
    p1b2(x));
  mk = mk >> ((10 - depth) * 3);
  mk = mk << ((10 - depth) * 3);
  return mk;
}

__device__ unsigned int getMKI(unsigned int x, unsigned int y,
    unsigned int z, unsigned int depth)
{
  return MK3(x, y, z, depth);
}

__device__ unsigned int getMK(double x, double y, double z, unsigned
    int depth, float sx, float sy, float sz)
{
  double dx = (double)x / sx;
  double dy = (double)y / sy;
  double dz = (double)z / sz;
  unsigned int uIx = (unsigned int)(dx * 0x400);
  unsigned int uIy = (unsigned int)(dy * 0x400);
  unsigned int uIz = (unsigned int)(dz * 0x400);
  return MK3(uIx, uIy, uIz, depth);
}

__device__ unsigned int getMKF(float x, float y, float z, unsigned int
    depth, float sx, float sy, float sz)
{
  double dx = (double)x;
  double dy = (double)y;
  double dz = (double)z;
  return getMK(dx, dy, dz, depth, sx, sy, sz);
}

__device__ unsigned int getMKN(double x, double y, double z, unsigned
```

```
    int depth)
{
  unsigned int uIx = (unsigned int)(x * 0x400);
  unsigned int uIy = (unsigned int)(y * 0x400);
  unsigned int uIz = (unsigned int)(z * 0x400);
  return MK3(uIx, uIy, uIz, depth);
}

__device__ unsigned int getMKNF(float x, float y, float z, unsigned
    int depth)
{
  double dx = (double)x;
  double dy = (double)y;
  double dz = (double)z;
  return getMKN(dx, dy, dz, depth);
}

__device__ unsigned int MKtoIndex(unsigned int mk, unsigned int depth)
{
  if (depth == 0)
    return 0;
  unsigned int valorTrpleta;
  valorTrpleta = mk << 2; //eliminamos el bit del padre
  valorTrpleta = valorTrpleta >> 2; //eliminamos el bit del padre
                    //valorTrpleta = valorTrpleta >> 32 - (depth * 3)
    ;//conservamos el valor de solo la cantidad de tripletas que
    ocuparemos
  valorTrpleta = valorTrpleta >> ((10 - depth) * 3);
  return valorTrpleta;
}

__device__ unsigned int IndextoMK(unsigned int index, unsigned int
    depth)
{
  if (index == 0)
    return 0x40000000;
  unsigned int mk = index << ((10 - depth) * 3);
  mk = mk | 0x40000000;
  return mk;
}

__device__ unsigned int MKtoIndexComplete(unsigned int mk, unsigned
```

```
    int depth)
{
  if (depth == 0)
    return 0;
  unsigned int offset = 0;
  for (unsigned int i = 1; i < depth; i++)
  {
    offset = offset + powArr[i]; //only pow base 8 from 0 to 9
  }

  unsigned int valorTrpleta;
  valorTrpleta = mk << 2; //eliminamos el bit del padre
  valorTrpleta = valorTrpleta >> 2; //eliminamos el bit del padre
  valorTrpleta = valorTrpleta >> ((10 - depth) * 3);
  return offset + valorTrpleta;
}

__device__ unsigned int IndextoMKComplete(unsigned int index, unsigned
    int depth)
{
  if (index == 0)
    return 0x40000000;
  unsigned int offset = 0, i = 1, morton = 0;
  while (offset <= index)
  {
    morton = index - offset;
    //offset = offset + pow(8, i);
    offset = offset + powArr[i]; //only pow base 8 from 0 to 9
    i++;
  }
  morton = morton << ((10 - depth) * 3);
  morton = morton | 0x40000000;
  return morton;
}

__device__ unsigned int getParentMK(unsigned int mk, unsigned int
   cLevel)
{
  return (mk >> ((10 - cLevel) * 3)) << ((10 - cLevel) * 3);
}

__device__ unsigned int getIntRep(float f)
```

```
129 {
130   double d = (double)f;
131   return (unsigned int)(d * 0x400);
132 }
133 
134 __device__ unsigned int getIntRep(double d)
135 {
136   return (unsigned int)(d * 0x400);
137 }
138 
139 __device__ float decodeMKX(unsigned int mk, unsigned int depth, float
      size)
140 {
141   unsigned int myMask = 7 << (10 - depth - 1) * 3;
142   mk = mk | myMask;
143   unsigned int res = c1b2(mk >> 0);
144   return (((float)res * size) / 0x400);
145 }
146 
147 __device__ float decodeMKY(unsigned int mk, unsigned int depth, float
      size)
148 {
149   unsigned int myMask = 7 << (10 - depth - 1) * 3;
150   mk = mk | myMask;
151   unsigned int res = c1b2(mk >> 1);
152   return (((float)res * size) / 0x400);
153 }
154 
155 __device__ float decodeMKZ(unsigned int mk, unsigned int depth, float
      size)
156 {
157   unsigned int myMask = 7 << (10 - depth - 1) * 3;
158   mk = mk | myMask;
159   unsigned int res = c1b2(mk >> 2);
160   return (((float)res * size) / 0x400);
161 }
```

Listing D.1: Funciones para construcción decodificación y manejo de claves Morton en CUDA.

Bibliografía

[1] M. Pidd y A. Carvalho, «Simulation software: not the same yesterday, today or forever», *Journal of Simulation*, vol. 1, n.º 1, págs. 7-20, 2006, ISSN: 1747-7778. DOI: 10.1057/palgrave.jos.4250004. dirección: https://doi.org/10.1057/palgrave.jos.4250004.

[2] E. Winsberg, *Science in the Age of Computer Simulation*. University of Chicago Press, 2013, ISBN: 9780226902043. DOI: 10.7208/chicago/9780226902050.001.0001. dirección: https://books.google.com.mx/books?id=goIi8auZ-hYC.

[3] P. J. N. Reddy, *Introduction to the Finite Element Method, Fourth Edition*, en, 4th edition. New York: McGraw-Hill Education, 2019, ISBN: 9781259861901. dirección: https://www.accessengineeringlibrary.com/content/book/9781259861901.

[4] K.-J. Bathe, «Finite Element Method», en *Wiley Encyclopedia of Computer Science and Engineering*. American Cancer Society, 2008, págs. 1-12, ISBN: 9780470050118. DOI: https://doi.org/10.1002/9780470050118.ecse159. dirección: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470050118.ecse159.

[5] G.-R. Liu, *Meshfree methods: moving beyond the finite element method*. CRC press, 2009.

[6] J. Kruger, P. Kipfer, P. Konclratieva y R. Westermann, «A particle system for interactive visualization of 3D flows», *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, n.º 6, págs. 744-756, 2005. DOI: 10.1109/TVCG.2005.87.

[7] O.-y. Song, D. Kim y H.-S. Ko, «Derivative particles for simulating detailed movements of fluids.», eng, *IEEE transactions on visualization and computer*

graphics, vol. 13, n.º 4, págs. 711-719, 2007, ISSN: 1077-2626 (Print). DOI: 10.1109/TVCG.2007.1022.

[8] M. Denneau, E. Kronstadt y G. Pfister, «Design and implementation of a software simulation engine», *Computer-Aided Design*, vol. 15, n.º 3, págs. 123-130, 1983, ISSN: 00104485. DOI: 10.1016/0010-4485(83)90077-5. dirección: https://www.sciencedirect.com/science/article/pii/0010448583900775.

[9] K. Popovici y P. J. Mosterman, *Real-Time Simulation Technologies: Principles, Methodologies, and Applications*, ép. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, 2017, ISBN: 9781439847237. dirección: https://books.google.com.mx/books?id=GvlzMPbuZqAC.

[10] J. J. Bartholdi y L. K. Platzman, «An O(N log N) planar travelling salesman heuristic based on spacefilling curves», *Operations Research Letters*, vol. 1, n.º 4, págs. 121-125, 1982, ISSN: 0167-6377. DOI: https://doi.org/10.1016/0167-6377(82)90012-8. dirección: https://www.sciencedirect.com/science/article/pii/0167637782900128.

[11] F. Zhang, J. Wu y X. Shen, «SPH-Based Fluid Simulation: A Survey», en *2011 International Conference on Virtual Reality and Visualization*, nov. de 2011, págs. 164-171. DOI: 10.1109/ICVRV.2011.18.

[12] M. Müller, D. Charypar y M. Gross, «Particle-Based Fluid Simulation for Interactive Applications», vol. 2003, jul. de 2003, págs. 154-159, ISBN: 1581136595.

[13] M. Macklin y M. Müller, «Position based fluids», *ACM Transactions on Graphics*, vol. 32, n.º 4, 104:1-104:12, jul. de 2013, ISSN: 07300301. DOI: 10.1145/2461912.2461984. dirección: http://doi.acm.org/10.1145/2461912.2461984.

[14] M. Müller, S. Schirm y M. Teschner, «Interactive blood simulation for virtual surgery based on smoothed particle hydrodynamics», *Technology and Health Care*, vol. 12, n.º 1, págs. 25-31, abr. de 2004, ISSN: 09287329. DOI: 10.3233/THC-2004-12103. dirección: http://dl.acm.org/citation.cfm?id=1011146.1011149%20https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/THC-2004-12103.

[15] M. MacKlin, M. Müller, N. Chentanez y T. Y. Kim, «Unified particle physics for real-time applications», *ACM Transactions on Graphics*, vol. 33, n.º 4, 153:1-153:12, jul. de 2014, ISSN: 15577333. DOI: 10.1145/2601097.2601152. dirección: http://doi.acm.org/10.1145/2601097.2601152.

[16] T. Takahashi, Y. Dobashi, T. Nishita y M. C. Lin, «An efficient hybrid incompressible SPH solver with interface handling for boundary conditions», *Computer Graphics Forum*, 2018, ISSN: 14678659. DOI: 10.1111/cgf.13292.

[17] C. Jun, Y. Kejian e Y. Yuan, «SPH-based visual simulation of fluid», en *Proceedings of 2009 4th International Conference on Computer Science and Education, ICCSE 2009*, jul. de 2009, págs. 690-693, ISBN: 9781424435210. DOI: 10.1109/ICCSE.2009.5228338.

[18] J. He, X. Chen, Z. Wang, K. Yan, C. Cao y Q. Peng, «A new adaptive model for real-time fluid simulation with complex boundaries», en *Computer-{Aided} {Design} and {Computer} {Graphics}, 2009. {CAD}/{Graphics} '09. 11th {IEEE} {International} {Conference} on*, ago. de 2009, págs. 45-48. DOI: 10.1109/CADCG.2009.5246934.

[19] J. Zhang, Y. Zhong y C. Gu, *Deformable Models for Surgical Simulation: A Survey*, 2018. DOI: 10.1109/RBME.2017.2773521.

[20] T. W. Sederberg y S. R. Parry, «Free-form deformation of solid geometric models», en *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986*, 1986, ISBN: 0897911962. DOI: 10.1145/15922.15903.

[21] Y. Duan, W. Huang, H. Chang, W. Chen, J. Zhou, S. Teo, Y. Su, C. K. Chui y S. Chang, «Volume preserved mass-spring model with novel constraints for soft tissue deformation», *IEEE Journal of Biomedical and Health Informatics*, 2016, ISSN: 21682194. DOI: 10.1109/JBHI.2014.2370059.

[22] S. Frisken, «3D ChainMail: a Fast Algorithm for Deforming Volumetric Objects», ago. de 2001.

[23] M. Müller, B. Heidelberger, M. Hennix y J. Ratcliff, «Position {Based} {Dynamics}», *3rd Workshop in Virtual Reality Interactions and Physical Simulation*, 2006.

[24] M. Freutel, H. Schmidt, L. Dürselen, A. Ignatius y F. Galbusera, «Finite element modeling of soft tissues: material models, tissue interaction and challenges.», eng, *Clinical biomechanics (Bristol, Avon)*, vol. 29, n.º 4, págs. 363-372, abr. de 2014, ISSN: 1879-1271 (Electronic). DOI: 10.1016/j.clinbiomech.2014.01.006.

[25] S. Cotin, H. Delingette y N. Ayache, «Real-time elastic deformations of soft tissues for surgery simulation», *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, n.º 1, págs. 62-73, ene. de 1999, ISSN: 1077-2626. DOI: 10.1109/2945.764872.

[26] S. De, J. Kim, Y. J. Lim y M. A. Srinivasan, «The point collocation-based method of finite spheres (PCMFS) for real time surgery simulation», *Computers and Structures*, vol. 83, n.º 17-18, págs. 1515-1525, 2005, ISSN: 00457949. DOI: 10.1016/j.compstruc.2004.12.003. dirección: https://www.sciencedirect.com/science/article/pii/S0045794905000799.

[27] S. De y K. J. Bathe, «The method of finite spheres», *Computational Mechanics*, vol. 25, n.º 4, págs. 329-345, 2000, ISSN: 01787675. DOI: 10.1007/s004660050481. dirección: https://doi.org/10.1007/s004660050481.

[28] M. Desbrun y M.-p. Gascuel, «Smoothed particles: A new approach for animating highly deformable bodies», *Computer Animation and Simulation*, vol. 96, ene. de 1996. DOI: 10.1007/978-3-7091-7486-9_5.

[29] A. Palyanov, S. Khayrulin y S. Larson, «Application of smoothed particle hydrodynamics to modeling mechanisms of biological tissue», *Advances in Engineering Software*, vol. 98, págs. 1-11, ago. de 2016. DOI: 10.1016/j.advengsoft.2016.03.002.

[30] J. Zhang, Y. Zhong, J. Smith y C. Gu, «Cellular neural network modelling of soft tissue dynamics for surgical simulation», *Technology and Health Care*, vol. 25, págs. 337-344, mayo de 2017. DOI: 10.3233/THC-171337.

[31] Epic Games, *Unreal Engine*, 2021. dirección: https://www.unrealengine.com.

[32] [. Technologies], *Unity*, 2021. dirección: https://unity.com/.

[33] B. T. Phong, «Illumination for Computer Generated Pictures», *Communications of the ACM*, vol. 18, n.º 6, págs. 311-317, jun. de 1975, ISSN: 15577317. DOI: 10.1145/360825.360839. dirección: https://doi.org/10.1145/360825.360839.

[34] [. Studio], *Microsoft Flight Simulator*, 2020. dirección: https://www.flightsimulator.com/.

[35] A. Alaraj, M. Lemole, J. Finkle, R. Yudkowsky, A. Wallace, C. Luciano, P. Banerjee, S. Rizzi y F. Charbel, «Virtual reality training in neurosurgery: Review of current status and future applications», *Surgical Neurology International*, vol. 2, n.º 1, pág. 52, 2011, ISSN: 2152-7806. DOI: 10.4103/2152-7806.80117. dirección: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3114314/.

[36] S. De, J. Kim y M. A. Srinivasan, «A meshless numerical technique for physically based real time medical simulations», *Studies in health technology and informatics*, vol. 81, págs. 113-118, 2001, ISSN: 0926-9630. dirección: http://europepmc.org/abstract/MED/11317723.

[37] Y.-J. Lim y S. De, «Real time simulation of nonlinear tissue response in virtual surgery using the point collocation-based method of finite spheres», *Computer Methods in Applied Mechanics and Engineering*, vol. 196, n.º 31, págs. 3011-3024, 2007, ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2006.05.015. dirección: http://www.sciencedirect.com/science/article/pii/S0045782507000369.

[38] S. Teodoro-Vite, J. S. Pérez-Lomelí, C. F. Domínguez-Velasco, A. F. Hernández-Valencia, M. A. Capurso-García y M. A. Padilla-Castañeda, «A High-Fidelity Hybrid Virtual Reality Simulator of Aneurysm Clipping Repair With Brain Sylvian Fissure Exploration for Vascular Neurosurgery Training», eng, *Simulation in Healthcare: The Journal of the Society for Simulation in Healthcare*, vol. Publish Ahead of Print, jul. de 2020, ISSN: 1559-2332. DOI: 10.1097/sih.0000000000000489.

[39] S. Tanaka, K. Harada, Y. Ida, K. Tomita, I. Kato, F. Arai, T. Ueta, Y. Noda, N. Sugita y M. Mitsuishi, «Quantitative assessment of manual and robotic microcannulation for eye surgery using new eye model», *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 11, n.º 2, págs. 210-217, 2015, ISSN: 1478-596X. DOI: 10.1002/rcs.1586. dirección: http://dx.doi.org/10.1002/rcs.1586.

[40] G. Azzie, J. T. Gerstle, A. Nasr, D. Lasko, J. Green, O. Henao, M. Farcas y A. Okrainec, «Development and validation of a pediatric laparoscopic surgery simulator», *Journal of Pediatric Surgery*, vol. 46, n.º 5, págs. 897-903, 2011, ISSN: 0022-3468. DOI: http://dx.doi.org/10.1016/j.jpedsurg.2011.02.026. dirección: http://www.sciencedirect.com/science/article/pii/S0022346811001382.

[41] N. Dal Ferro, A. G. Strozzi, C. Duwig, P. Delmas, P. Charrier y F. Morari, «Application of smoothed particle hydrodynamics (SPH) and pore morphologic model to predict saturated water conductivity from X-ray CT imaging in a silty loam Cambisol», *Geoderma*, vol. 255-256, págs. 27-34, 2015, ISSN: 0016-7061. DOI: https://doi.org/10.1016/j.geoderma.2015.04.019. dirección: https://www.sciencedirect.com/science/article/pii/S0016706115001330.

[42] W. Fan y A. Bifet, «Mining Big Data: Current Status, and Forecast to the Future», *SIGKDD Explor. Newsl.*, vol. 14, n.º 2, págs. 1-5, abr. de 2013, ISSN: 1931-0145. DOI: 10.1145/2481244.2481246. dirección: https://doi.org/10.1145/2481244.2481246.

[43] D. J. Hand y N. M. Adams, «Data Mining», en *Wiley StatsRef: Statistics Reference Online*. American Cancer Society, 2015, págs. 1-7, ISBN: 9781118445112. DOI: https://doi.org/10.1002/9781118445112.stat06466.pub2. dirección: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat06466.pub2.

[44] D. Soriano-Valdez, I. Pelaez-Ballestas, A. Manrique de Lara y A. Gastelum-Strozzi, «The basics of data, big data, and machine learning in clinical practice», *Clinical Rheumatology*, 2020, ISSN: 1434-9949. DOI: 10.1007/s10067-020-05196-z. dirección: https://doi.org/10.1007/s10067-020-05196-z.

[45] D. A. S. Valdez, P. Delmas, T. Gee, P. Gutierrez, J. L. Punzo-Diaz, R. Ababou y A. G. Strozzi, «CUDA Implementation of a Point Cloud Shape Descriptor Method for Archaeological Studies», en *Advanced Concepts for Intelligent Vision Systems*, J. Blanc-Talon, P. Delmas, W. Philips, D. Popescu y P. Scheunders, eds., Cham: Springer International Publishing, 2020, págs. 457-466, ISBN: 978-3-030-40605-9.

[46] Microsoft, *Microsoft Visual Studio*, 2016. dirección: https://visualstudio.microsoft.com.

[47] NVIDIA, P. Vingelmann y F. H. P. Fitzek, *CUDA, release: 10.2.89*, 2020. dirección: https://developer.nvidia.com/cuda-toolkit.

[48] H. Samet, «Neighbor finding in images represented by octrees», *Computer Vision, Graphics, and Image Processing*, vol. 46, n.º 3, págs. 367-386, 1989, ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(89)90038-8. dirección: http://www.sciencedirect.com/science/article/pii/0734189X89900388.

[49] X. Zhao, F. Li y S. Zhan, «A new GPU-based neighbor search algorithm for fluid simulations», en *2010 2nd International Workshop on Database Technology and Applications, DBTA2010 - Proceedings*, nov. de 2010, págs. 1-4, ISBN: 9781424469772. DOI: 10.1109/DBTA.2010.5659015.

[50] J. Behley, V. Steinhage y A. B. Cremers, «Efficient radius neighbor search in three-dimensional point clouds», en *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, mayo de 2015, págs. 3625-3630. DOI: 10.1109/ICRA.2015.7139702.

[51] M. Joselli, J. R. Da y E. Clua, «NGrid: A proximity data structure for fluids animation with GPU computing», en *Proceedings of the ACM Symposium on Applied Computing*, ép. SAC '15, vol. 13-17-April-2015, New York, NY, USA: ACM, 2015, págs. 1303-1308, ISBN: 9781450331968. DOI: 10.1145/2695664.2695827. dirección: http://doi.acm.org.pbidi.unam.mx:8080/10.1145/2695664.2695827.

[52] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*. Ottawa: International Business Machines Co., 1966.

[53] N. Satish, M. Harris y M. Garland, «Designing efficient sorting algorithms for manycore GPUs», en *2009 IEEE International Symposium on Parallel Distributed Processing*, 2009, págs. 1-10. DOI: 10.1109/IPDPS.2009.5161005.

[54] W.-m. W. Hwu, *GPU Computing Gems Jade Edition*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011, ISBN: 9780123859631.

[55] J. J. Monaghan, «Smoothed particle hydrodynamics», *{\textbackslash}araa*, vol. 30, págs. 543-574, 1992. DOI: 10.1146/annurev.aa.30.090192.002551.

[56] M. B. Liu y G. R. Liu, «Smoothed particle hydrodynamics (SPH): An overview and recent developments», *Archives of Computational Methods in Engineering*, vol. 17, n.º 1, págs. 25-76, 2010, ISSN: 11343060. DOI: 10.1007/s11831-010-9040-7. dirección: http://dx.doi.org/10.1007/s11831-010-9040-7.

[57] M. L. GR Liu, *Smoothed Particle Hydodynamics: a meshfree particle method*. Worls Scientific, 2005, pág. 448.

[58] J. P. Morris, «Analysis of smoothed particle hydrodynamics with applications», Tesis doct., 1996.

[59] C. Pozrikidis, *Fluid Dynamics: Theory, Computation, and Numerical Simulation*, 3rd. Springer Publishing Company, Incorporated, 2016, ISBN: 1489979905.

[60] S. Li y W. Wang, «Water pouring effect simulation based on SPH», en *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, dic. de 2016, págs. 94-97. DOI: 10.1109/ICCSNT.2016.8070126.

[61] X. Shao, B. Wang y M. Zhao, «Unified particle-based coupling algorithm for stable soft tissue-blood interaction», en *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, nov. de 2016, págs. 1-6. DOI: 10.1109/ICARCV.2016.7838659.

[62] P. W. Cleary y R. Das, «The Potential for SPH Modelling of Solid Deformation and Fracture», en *IUTAM Symposium on Theoretical, Computational and Modelling Aspects of Inelastic Media*, B. D. Reddy, ed., Dordrecht: Springer Netherlands, 2008, págs. 287-296, ISBN: 978-1-4020-9090-5.

[63] J. J. Monaghan, «SPH without a Tensile Instability», *Journal of Computational Physics*, vol. 159, n.º 2, págs. 290-311, 2000, ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.2000.6439. dirección: https://www.sciencedirect.com/science/article/pii/S0021999100964398.

[64] J. P. Gray, J. J. Monaghan y R. P. Swift, «SPH elastic dynamics», *Computer Methods in Applied Mechanics and Engineering*, vol. 190, n.º 49, págs. 6641-6662, 2001, ISSN: 0045-7825. DOI: https://doi.org/10.1016/S0045-7825(01)00254-7. dirección: https://www.sciencedirect.com/science/article/pii/S0045782501002547.

[65] C. Tu y L. Yu, «Research on Collision Detection Algorithm Based on AABB-OBB Bounding Volume», en *2009 First International Workshop on Education Technology and Computer Science*, vol. 1, 2009, págs. 331-333, ISBN: VO - 1. DOI: 10.1109/ETCS.2009.82.

[66] H. A. Sulaiman, M. A. Othman, M. M. Ismail, M. A. M. Said, A. Bade y M. H. Abdullah, «Methodology of performing narrow phase collision detection for virtual environment», en *2014 International Symposium on Technology Management and Emerging Technologies*, 2014, págs. 511-515, ISBN: VO -. DOI: 10.1109/ISTMET.2014.6936564.

[67] B. Yong, J. Shen, H. Sun, H. Chen y Q. Zhou, «Parallel GPU-based collision detection of irregular vessel wall for massive particles», *Cluster Computing*, vol. 20, n.º 3, págs. 2591-2603, 2017, ISSN: 1573-7543. DOI: 10.1007/s10586-017-0741-7. dirección: https://doi.org/10.1007/s10586-017-0741-7.

[68] L. TANG, W.-g. SONG, T.-c. HOU, L.-l. LIU, W.-x. CAO e Y. ZHU, «Collision detection of virtual plant based on bounding volume hierarchy: A case study on virtual wheat», *Journal of Integrative Agriculture*, vol. 17, n.º 2, págs. 306-314, 2018, ISSN: 2095-3119. DOI: https://doi.org/10.1016/S2095-3119(17)61769-6. dirección: http://www.sciencedirect.com/science/article/pii/S2095311917617696.

[69] A. Kolb, L. Latta y C. Rezk Salama, *Hardware-based Simulation and Collision Detection for Large Particle Systems*. ene. de 2004. DOI: 10.1145/1058129.1058147.

[70] X. Wu, X. Zhu, G.-Q. Wu y W. Ding, «Data mining with big data», *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, n.º 1, págs. 97-107, 2014. DOI: 10.1109/TKDE.2013.109.

[71] A. L'Heureux, K. Grolinger, H. F. Elyamany y M. A. M. Capretz, «Machine Learning With Big Data: Challenges and Approaches», *IEEE Access*, vol. 5, págs. 7776-7797, 2017. DOI: 10.1109/ACCESS.2017.2696365.

[72] Blender Online Community, *Blender - a 3D modelling and rendering package. URL: http://www.blender.org/*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2013. dirección: http://www.blender.org/.

[73] P. Min, *binvox*, http://www.patrickmin.com/binvox.

[74] R. Manning, J. P. Griffith, T. F. Pigot y L. F. Vernon-Harcourt, *On the flow of water in open channels and pipes*. 1890.