UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACION
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS
SEÑALES, IMÁGENES Y AMBIENTES VIRTUALES

SIMULACIÓN DE FLUJOS SANGUÍNEOS Y DEFORMACIÓN DE VASOS SANGUÍNEOS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
V I L L A L D A Q U E Z A D A M A R C O A N T O N I O

DIRECTOR DE TESIS:
DR. FERNANDO ARÁMBULA COSÍO
Centro de Ciencias Aplicadas y Desarrollo Tecnológico

CODIRECTOR DE TESIS:
DR. ALFONSO GASTÉLUM STROZZI
Centro de Ciencias Aplicadas y Desarrollo Tecnológico

Ciudad Universitaria, CD. MX. ENERO 2017

A mis padres y tutores que me soportaron y apoyaron a lo largo del camino, sin esperar nada a cambio, soportando mis locuras y brindándome un espacio y tiempo para trabajar, permitiéndome desarrollar una loca idea que se materializo en la presente tesis.
Muchas Gracias!!

Contents

Lista de Imágenes

Lista de Tablas

Chapter 1

Introducción

Fluido es el termino referido a sustancias líquidas o gaseosas que están en constante movimiento, estos pueden ser vistos como un conjunto de elementos que tienen su propia trayectoria, no estan atados entre sí, y sin embargo cada uno de los elementos que conforman el fluido se comportan de manera similar, al seguir la trayectoria de menor resistencia. Existen diferentes propuestas en las últimas décadas que han presentado formas de modelar la naturaleza de los fluidos (1, 2, 3, 4, 5, 6), desde el movimiento de las olas (7), el flujo de aire que pasa por el ala de un avión, la absorción de fluidos en lo suelos, el movimiento del flujo volcánico (8), hasta el modelado de fenómenos astronómicos (9), con el único propósito de modelar un fluido en específico, estos aunque diferentes en naturaleza, comparten similitudes que permiten una comparación en la forma en que son modelados. El presente trabajo tiene como propósito describir la implementación del método SPH (Smooth Particle Hydrodinamics), implementada en tarjetas gráficas, con el objetivo de simular de manera particular el fluido biológico de la sangre a través de los vasos sanguíneos, así mismo esta implementación utiliza un sistema distribuido heterogéneo de máquinas que poseen tarjetas gráficas, las cuales pueden presentar características distintas, esto se realiza con el propósito de acelerar los cálculos del fluido dentro de la simulación.

El método SPH es un método de interpolación Lagrangiano, el cual permite calcular el movimiento de un fluido al representarlo mediante un conjunto de partículas (9), de esta manera el movimiento de un fluido es aproximado por el movimiento de las partículas que lo representan. Para predecir este movimiento, en cada iteración todas las posiciones de las partículas deben ser calculadas antes del siguiente paso de la simulación. Estas nuevas posiciones son calculadas aplicando a cada una de las partículas el mismo conjunto de ecuaciones gobernantes, sin embargo el método no requiere de un orden específico para procesar cada una de las partículas que representan al fluido, solo se requiere que todas las partículas sean procesadas antes de la siguiente iteración, así mismo las ecuaciones gobernantes aplicadas no dependen de la siguiente o anterior posición de alguna partícula, solo de las posiciones actuales de las partículas vecinas, respecto a la partícula que se está procesando. Debido a estas características el cálculo de cada nueva posición puede ser calculada de manera independiente, esta es una de las

principales ventajas que presenta el método para ser implementado en Tarjetas Gráficas de Propósito General (GPGPU) ya que el cálculo de las nuevas posiciones puede ser reducido de manera significativa, así mismo la propuesta presentada para localizar todas las partículas vecinas de cada una de las partículas de la simulación, mediante el uso de regiones vecinas, permite la utilización de un sistema distribuido de computadoras, al reducir el número de partículas que deben compartirse entre las distintas tarjetas gráficas.

El uso de tarjetas gráficas para calcular las nuevas posiciones de las partículas, permite reducir los tiempos de cálculo significativamente, sin embargo la limitada memoria que actualmente poseen estas, no permite el procesamiento de un gran número de partículas a la vez (Apéndice A), esta cantidad de memoria si bien ha sido aumentada de manera continua en los últimos años, no esta a la par con la necesidad de procesar los miles de partículas a la vez, necesarias para simular un fluido de tamaño significativo (1 millón de partículas o más), esto significa que si se desea el procesamiento de las partículas con el uso de las tarjetas gráficas es necesario copiar las partículas de la memoria de la CPU a la memoria de la GPU, procesar este grupo de partículas, copiar los resultados de la GPU y almacenarlos nuevamente en la CPU, repitiendo este proceso hasta terminar de evaluar todas las partículas involucradas en la simulación, este coste computacional que es adicional al método SPH, incrementa los tiempos de la simulación, pero no al punto de ser mas lento que la implementación del método sin la ayuda de las tarjetas gráficas (10), así mismo con el propósito de reducir aún mas los tiempos de cálculo, se planteo el desarrollo del simulador haciendo uso de un sistema distribuido de máquinas, permitiendo mantener las partículas en la memoria de las tarjetas gráficas el mayor tiempo posible, y reduciendo la cantidad datos de transferencia entre la GPU y la CPU , al limitar el envió de las partículas que se encuentran en la frontera de dos máquinas, las cuales pueden interactuar con otras partículas de una o varias tarjetas gráficas, ya sea que estas se encuentren en la misma máquina o en otra máquina, que puede o no encontrarse en la misma red local, esta implementación distribuida bajo condiciones especificas, muestra una mejora en los tiempos de cálculo, al permitir un mayor número de procesamiento de partículas a la vez.

1.1 Relevancia y contribución del trabajo

Este trabajo tiene como propósito desarrollar un sistema de simulación de fluidos, considerando como caso de estudio el movimiento del flujo sanguíneo a través de vasos arteriales, utilizando mediciones biológicas para afectar los parámetros de la simulación, como la edad para determinar la flexibilidad de las paredes arteriales, y factores que modifican la viscosidad de la sangre como es la forma de los glóbulos rojos que es aproximada por los niveles de sales y proteínas en el plasma, que conforman la sangre. De esta forma uno de los objetivos planteados es estimar la interacción del flujo sanguíneo con objetos ajenos a este, como es el caso de la introducción de un catéter a través del torrente sanguíneo o la interacción de un dispositivo que permita incrementar el flujo sanguíneo como es el uso de un stent, el primero interactúa con el flujo sanguíneo de manera temporal mientras que el segundo es colocado de forma permanente, en este último caso surge la pregunta acerca de los posibles efectos que el paciente podría desarrollar a lo largo del tiempo, desafortunadamente no existe otra forma de comprobar estos efectos más que de manera empírica, mediante el estudio de personas que tienen colocado un dispositivo stent, sin embargo en la mayoría de los casos estas personas no presentan las mismas características como son: hábitos alimenticios, ubicación del stent, edad y problemas vasculares, lo que hace que las posibles complicaciones sean representadas en porcentajes respecto a los casos mas similares (11), lo que hace que el diseño de nuevos dispositivos sea una tarea por demás difícil, al tener que validar la introducción de nuevos diseños mediante el uso en estudios clínicos para su uso comercial (12), sólo después de presentar una serie de regulaciones para su posible uso en seres humanos (13), esta posible utilización del simulador no esta contemplada en el presente trabajo, debido a los muchos parámetros que se necesitan validar, para poder considerar validar un nuevo diseño de stents, y es planteado para un alcance a futuro, debido a que ocupa menos recursos, el diseño y validación de estos. El alcance actual del simulador solo presenta estimaciones de como objetos ajenos al fluido sanguíneo interactúan con este.

Actualmente existen una gran variedad de trabajos enfocados en la simulación de fluidos biológicos, en el caso de la simulación del flujo sanguíneo, la mayoría de estos trabajos, encontrados en la literatura solucionan el problema desde el punto de vista Euleriano (14, 15, 16) y resolviendo una sola problemática en específico, así mismo estas soluciones no permiten la interacción de objetos con el fluido, ya que estas interacciones son consideradas como parte de la frontera del fluido.

1.2 Metas propuestas y alcanzadas

Aunque han surgido nuevos trabajos en los últimos años que hacen uso del método SPH para simular fluidos, la principal contribución de este trabajo sera la implementación del método SPH en paralelo, en un sistema distribuido de máquinas que utiliza las tarjetas gráficas de distintas máquinas que no necesariamente tienen las mismas características,

con el objetivo de acelerar los cálculos de la simulación, aproximando de forma mas rápida el flujo sanguíneo a través de los vasos sanguíneos en movimiento, así mismo se presenta una propuesta para estimar la interacción de objetos ajenos al movimiento del flujo sanguíneo.

Es necesario mencionar que parte de la problemática de la implementación de la simulación que hace uso de las tarjetas gráficas, fue la búsqueda de los vecinos de cada una de las partículas, en pruebas iniciales se contemplo el uso de listas enlazadas y del uso de estructuras de ordenación como es el caso de arboles binarios de búsqueda, sin embargo se descartaron estas ideas debido al movimiento de las partículas, que requería reorganizar la información en cada iteración, con varias condiciones de carrera, lo cual en comparación con la CPU se realizaba de manera lenta, dentro de la tarjeta gráfica, también se descarto debido al uso de grandes bloques de memoria para organizar y definir estructura de datos, es por ello que se utilizo una nueva propuesta que involucra la organización de las partículas en sectores, de esta forma se acelera la búsqueda de las partículas vecinas, y a la vez esta propuesta permite evitar condiciones de carrera al momento de repartir el trabajo en cada uno de los threads dentro de la tarjeta gráfica (Apéndice C), así mismo esta propuesta permite solucionar el problema de reordenación de las partículas, que presentan las estructuras de datos previamente mencionadas, al ordenar en cada iteración los sectores mediante el uso de algoritmos de ordenación, optimizados para su uso en las tarjetas gráficas.

Además de la forma de organización propuesta se implemento un algoritmo que permite unir dos listas ordenas de manera óptima en paralelo, de esta forma podemos realizar la ordenación de las partículas en sectores de forma óptima, utilizando el algoritmo de ordenación Bitonic Sort (17), sin la limitante de requerir que el número de partículas sea igual a una potencia dos, esto se detalla con mas precisión en el capítulo 4. Esta necesidad de implementar una nueva forma de ordenación surge de la necesidad de utilizar una forma alterna a los métodos de ordenación proporcionados por la API de CUDA Thrust, que demostraron ser poco confiables, debido a que se encontró que al ordenar de manera correcta varias partículas en sus correspondientes sectores (más de 10, 000 partículas), después de varias iteraciones de simulación, ocurría un error de memoria, al no liberar correctamente la memoria auxiliar utilizada para la ordenación de estas.

1.3 Presentación del trabajo

La tesis desarrollada es dividida de la siguiente forma:

- En el segundo capítulo se presenta el caso de estudio propuesto, explicando algunos conceptos utilizados para aproximar de manera casi realista el fluido que se pretende simular, exponiendo los componentes de la sangre, la dilatación y contracción de las paredes arteriales como respuesta a estímulos externos, así como algunas simplificaciones que se consideran para simplificar la simulación.
- En el tercer capítulo se presenta una introducción a la mecánica de fluidos, explicando superficialmente los métodos Eulerianos y Lagrangianos marcando las diferencias entre estos, así mismo se muestran las características, ventajas y limitantes de la implementación Lagrangiana elegida para la simulación del flujo sanguíneo.
- En el cuarto capítulo se expone la implementación del método SPH resaltando las limitantes de memoria por parte del uso de las tarjetas gráficas, así mismo se presentan posibles escenarios de implementación considerando la capacidad de memoria y tiempos de procesamiento por parte de estas, en este capítulo también se presenta una justificación de la implementación del sistema distribuido, mostrando las limitantes de la red y los posibles errores que pueden suscitar su uso.
- En el quinto capítulo se muestran los resultados del caso de estudio y los métodos utilizados para validar la implementación del método SPH.
- Finalmente en el sexto capítulo se presentan las conclusiones y los posibles alcances del simulador, así como las posibles mejoras que se podrían realizarse en un futuro.

Chapter 2

Caso de Estudio

Un organismo unicelular puede ser reducido como un elemento que absorbe los nutrientes necesarios para mantenerse vivo, desechando los subproductos que no utiliza, contaminando su medio circundante, esto continua tanto como su entorno se lo permita. Algo similar ocurre dentro de nuestro cuerpo, ignorando los complicados procesos realizados dentro de este, podemos reducir las células que nos componen a elementos que absorben y desechan productos en su entorno inmediato, con la diferencia que algunos de estos productos de desecho son utilizados por otras células, manteniendo un equilibrio en el cuerpo humano, esto se logra al mover los subproductos de desecho a las células que utilizarán estos subproductos para ser metabolizados o desechados del cuerpo, manteniendo un ambiente óptimo para todas las células del cuerpo humano, esto es logrado mediante el sistema circulatorio, el cual transporta nutrientes y desechos a lo largo del cuerpo humano, es por ello que el sistema circulatorio es uno de los principales temas de estudio en las últimas décadas (18, 19).

Al entender la problemática que se pretende simular se deja de lado cualquier variable que pueda afectar la exactitud de la simulación, es por ello que primero se debe entender la estructura que mantiene la sangre en circulación, por esta razón se presenta una breve introducción del sistema circulatorio, y aunque el presente trabajo no se tiene como objetivo el simular todo el sistema circulatorio, en necesario entender como es afectado los segmentos arteriales que son propuestos como caso de estudio.

2.1 Sistema Circulatorio

El objetivo del sistema circulatorio es transportar las sustancias requeridas para las células como el oxígeno, hormonas y vitaminas removiendo subproductos a lo largo del cuerpo humano, así mismo este es uno de los principales mecanismos encargado de regular la temperatura en el cuerpo humano enviando calor del centro del cuerpo a la periferia de este, donde será disipado por el ambiente. Su importancia es indispensable para mantener el cuerpo humano en condiciones óptimas, si una parte de este deja de funcionar correctamente, puede dar lugar a fallas en los órganos conectados a este,

como lo son el cerebro, corazón o pulmones.

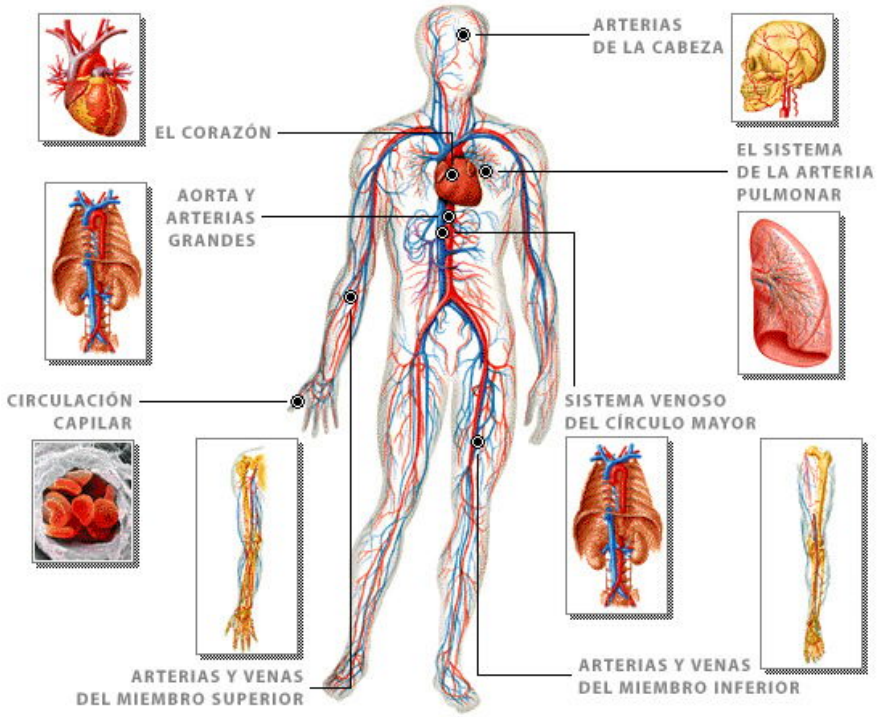

Figure 2.1: Representación del sistema circulatorio

El sistema circulatorio puede ser simplificado por los siguientes elementos:

- El corazón que impulsa el flujo sanguíneo,
- Los vasos sanguíneos que recorre y conecta todo el cuerpo humano, y
- La sangre como medio de transporte que contiene los nutrientes y desechos que son metabolizados.

De esta manera el sistema circulatorio transporta oxígeno para los tejidos y órganos, recolectando los productos de desecho y transportando estos, a los órganos encargados de su correcta eliminación, manteniendo en funcionamiento el cuerpo humano.

El presente trabajo no contempla la simulación de todo el sistema circulatorio, solo una parte de este, en especifico una parte de los vasos sanguíneos, es por ello que se deja de lado la modelación del corazón, el paso por los pulmones, así como el mecanismo de liberación de gases en los capilares, el principal estudio se realiza en las arterias, sin embargo es necesario entender el funcionamiento de todo el sistema circulatorio para comprender las problemáticas que se enfrentan al simular una sola parte de este.

2.1.1 El Corazón como Mecanismo de Bombeo

El corazón es la bomba que impulsa la sangre a través del cuerpo humano en la circulación pulmonar y sistémica, el corazón es dividido en dos lados, el lado derecho que transporta la sangre a los pulmones (sistema pulmonar) y el lado izquierdo que transporta la sangre al resto del cuerpo humano (sistema sistémico), cada lado es a la vez dividido en dos cámaras, en una aurícula y un ventrículo. Los ventrículos son las principales cámaras de bombeo en el corazón, mientras que las aurículas funcionan como cámaras de recolección para la sangre que es recibida por parte del sistema pulmonar y sistémico. La conexión de la aurícula al ventrículo ya sea el lado izquierdo o derecho, están conectados de manera directa y solo se encuentra separados por una válvula que regula la entrada de sangre al ventrículo, mientras que la conexión del ventrículo a la aurícula de lado opuesto, están conectadas de manera indirecta por medio del sistema pulmonar (ventrículo derecho a la aurícula izquierda) y del sistema sistémico (ventrículo izquierdo a la aurícula derecha), y son estas cuatro cámaras las principales fuentes mecánicas encargadas de generar la fuerza necesaria para mantener la sangre en movimiento.

Cada una de las cámaras que forman el corazón esta conectada en serie, la entrada de una cámara es la salida de otra cámara, ya sea de manera directa o indirecta, la sangre que regresa de la circulación sistémica, provee los nutrientes al cuerpo, esta entra al lado derecho del corazón por las venas, y bajo condiciones normales la sangre que entra a la aurícula derecha se encuentra desoxigenada, debido a que se libero el oxígeno recolectado de los pulmones a lo largo del cuerpo humano, esta sangre desoxigenada, una vez que entra al corazón es enviada nuevamente a los pulmones donde recolecta nuevamente oxígeno y libera el dióxido de carbono recolectado, al terminar este proceso entra al lado izquierdo del corazón para nuevamente recorrer el cuerpo a través de las arterias (figura 2.2).

La acción de bombeo del corazón puede ser dividido en dos fases: la fase de contracción que empuja la sangre para salir del corazón a través de las arterias (fase sistólica), y la fase de dilatación donde las venas llenan de sangre la aurícula derecha del corazón (fase diastólica). Durante la fase sistólica la presión de la aorta, ubicada a la salida del ventrículo izquierdo del corazón, se eleva de 12 mm Hg a 25 mm Hg de presión, al termino de la fase sistólica la válvula que se conecta con esta se cierra y la presión intraventricular cae abruptamente, con el objetivo de evitar el regreso de la sangre al corazón, es aquí donde la aorta y las arterias musculares ayudan al sistema circulatorio a mantener el flujo sanguíneo en un solo sentido.

No existe un dispositivo de bombeo creado hasta el momento que pueda ser igualado al corazón, esto lo podemos ver al considerar que el tiempo promedio de vida para un ser humano es de 70 años, y con un promedio de 70 latidos por minuto, podemos estimar que este producirá al menos $2.6\text{x}10^{10}$ latidos antes de detenerse, una maravilla de la naturaleza que opaca los demás elementos que constituyen el sistema circulatorio.

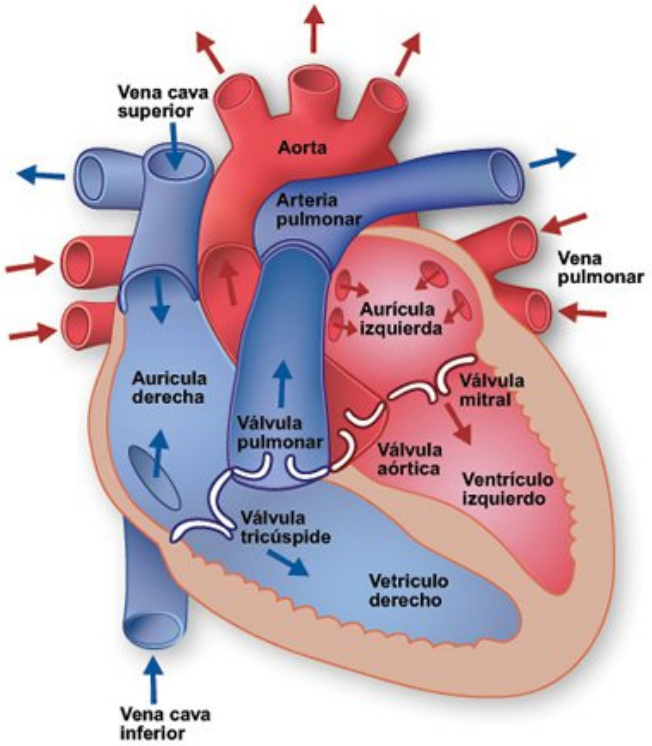

Figure 2.2: Representación de las cámaras internas del corazón

2.1.2 Vasos Sanguíneos

Los diferentes vasos sanguíneos en el sistema circulatorio están formados por diferentes estructuras y por lo tanto poseen propiedades fisiológicas diferentes, debido a que el presente trabajo tiene como objetivo simular el flujo sanguíneo dentro de las arterias a continuación se presenta de manera superficial los diferentes tipos de vasos sanguíneos.

La morfología básica de las paredes arteriales es mostrada en la figura 2.3, la capa mas interior de los vasos sanguíneos es el endotelio, esta capa cubre la pared interior y esta presente a lo largo de las arterias, venas y capilares. Además del endotelio los vasos sanguíneos están compuestas por diferentes laminas que dependiendo de su composición (diámetro y la distancia del corazón) tendrán una función diferente. En particular la clasificación de las arterias pueden ser divididas como arterias elásticas, arterias musculares o arteriolas.

- Las arterias elásticas son relativamente grandes y están compuesta por varias capas de escleroproteinas (elastina), mientras las paredes que la componen son relativamente delgadas, un ejemplo de una arteria elástica es la aorta donde el área transversal (lumen), el espacio que encierra la pared arterial es mayor que el ancho de las paredes, lo que hace que las paredes sean consideradas delgadas en

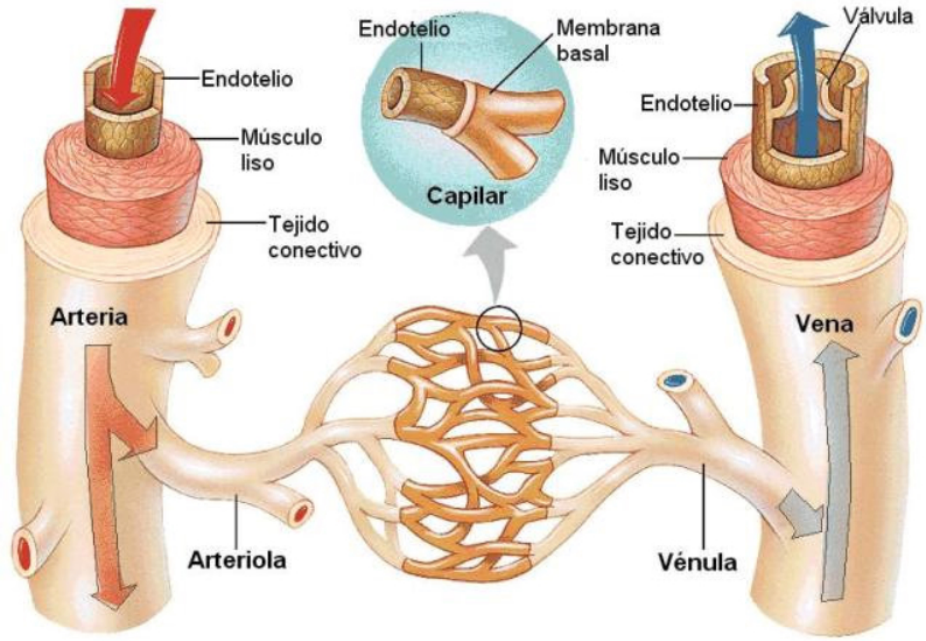

Figure 2.3: Representación de la estructura básica de los vasos sanguíneos

comparación con las arterias musculares las cuales requieren su propio suministro de sangre.

- Las arterias musculares por otra parte son los vasos sanguíneos intermedios clasificadas entre las grandes arterias elásticas y las pequeñas arteriolas, siendo estas las mas numerosas encontradas en el sistema circulatorio. Las paredes de las arterias musculares como su nombre lo indica están formadas por un capa de fibras musculares, estas capas están rodeadas por una capa de componentes conectivos, lo que permite el cambio del diámetro de las arterias casi de manera inmediata.

- Las arteriolas son las arterias mas pequeñas con un diámetro de 0.3 mm o menor, rara vez contienen membranas elásticas y están compuestas por fibras conectivas y fibras de colágeno.

La transición de un tipo de arteria a otro tipo, puede ser abrupto o suave, y sin embargo el flujo de sangre es mantenido suave, constante y en una sola dirección gracias a las arterias musculares y la capa endotelial que es continua, bajo condiciones normales. Además del cambio del diámetro en las estructura de las arterias se debe considerar las contracciones del corazón que expulsa la sangre oxigenada a intervalos y no como un flujo constante, si no se consideraran las paredes de los vasos sanguíneos flexibles, el flujo sanguíneo llegaría en los capilares en forma errática, lo que no permitirá el intercambio de sustancias en los capilares, afortunadamente las arterias en condiciones normales son elásticas, lo que permiten contraerse y dilatarse para mantener un flujo

de sangre constante en una sola dirección, por lo que pueden considerarse a estos como un mecanismo auxiliar de bombeo para llevar de regreso la sangre al corazón.

Otro elemento a tomar en cuenta son los barorreceptores, localizados en la túnica adventicia, estos son encargados de detectar los cambios de presión dentro de la arteria y por lo tanto de promover la dilatación o contracción de las arterias, este elemento es importante para la simulación ya que podemos utilizarlos como puntos de control distribuidos a lo largo del segmento de arteria simulado, aprovechando además que estos son independientes entre sí.

El presente trabajo no considera la simulación de la venas debido a que estas tienen válvulas que evitan que el flujo sanguíneo regrese por ellas, y aunque las venas son mas abundantes en el cuerpo humano que las arterias, presentan mas problemas para su simulación como lo es: paredes mas delgadas y diámetros mayores, asi mismo las venas son mucho menos elásticas y tienden a colapsar mas rápidamente, sin mencionar que las propiedades musculares y elásticas de estas son casi inexistente, estas características presentan sus propias ventajas y desventajas en el planteamiento del problema, es por ello que se decidió la simulación de segmentos de arterias musculares.

Vasos Arteriales	Largo Promedio (cm)	Diametro (cm)	Ancho de la Pared (cm)	Velocidad Promedio (cm/s)
Capilares	0.0008	0.1	0.0001	0.1
Venulas	0.002	0.2	0.0002	0.2
Arteriolas	0.005	1	0.02	5
Arterias	0.4	50	0.1	45
Venas	0.5	2.5	0.05	1.0
Aorta	2.5	50	0.2	48
Vena Cava	3.0	50	0.15	38

Table 2.1: Resumen de la aproximación de los vasos sanguíneos

2.1.3 Sangre como medio de transporte

La sangre es el principal medio de transporte en el cuerpo humano, este transporta gases, nutrientes, y productos de desecho resultantes de procesos metabólicos. Este fluido biológico es una suspensión acuosa, no homogénea que posee características inusuales debido a su composición. Dentro de las arterias en condiciones normales la sangre se encuentra en constante movimiento y esta compuesta por distintos tipos de elementos suspendidos en una solución acuosa, llamada plasma, el plasma por si

mismo contiene un espectro complejo de moléculas orgánicas, pero esta mayormente compuesto por agua, además del plasma la sangre esta principalmente compuesta por glóbulos rojos (RBC o Eritrocitos), células blancas (WBC White Blood Cells) y plaquetas (trombositos), de estos elementos los glóbulos rojos constituyen el 95% de la sangre, por lo que son los glóbulos rojos los que juegan un importante rol en la definición de las propiedades mecánicas de la sangre.

Elementos	Proporciones Relativas
Celulas Rojas (Eritrocitos)	6000
Celulas Blancas (Leucocitos)	1
Plaquetas (Trombocitos)	30
Plasma	**Fraccion de Peso**
Agua	0.91
Proteinas	0.07
Solutos Inorgánicos	0.01
Otras sustancias Organicas	0.01

Table 2.2: Constitución de la Sangre ($5x10^6$ particulas / mm^3)

Debido a que los glóbulos rojos componen la mayor parte de la sangre, es necesario estudiar las características de estos para entender algunas de las propiedades de la sangre, una de estas propiedades es el poseer, una membrana flexible que encierran una solución concentrada de hemoglobina, la viscosidad de la hemoglobina es 5 veces mayor que la sangre, lo que hace posible que los glóbulos rojos puedan deformarse, esta propiedad permite que los glóbulos rojos de un promedio de 8 μm no sólo pase por los capilares de 5 μm, sino también por las capas endoteliales que cubren los vasos sanguíneos.

Otra característica que tienen las células rojas es que tienden a agruparse y formar una estructura continuas cuando esta en reposo, llamadas rouleaux, o bien agrupaciones dentro de la suspensión de la sangre, como se muestra en la figura 2.4.

Asi mismo una sedimentación ocurre cuando la estructura de las células rojas se rompe bajo una presión excesiva, después de este rompimiento estas se convierten en una suspensión que se agrega al plasma en el que se encuentran, o bien si no existe suficiente presión estas formar agregados que pueden convertirse en pequeñas hileras llamadas rouleaux. Existe un equilibrio dinámico entre el tamaño de los agregados y los rouleaux dependiendo de la orientación de los glóbulos rojos que esta en relación a la presión aplicada, bajo la correcta presión los agregados y los reoleaux son reducidos a células individuales. Por lo que la relación del estrés cortante y la formación de estos

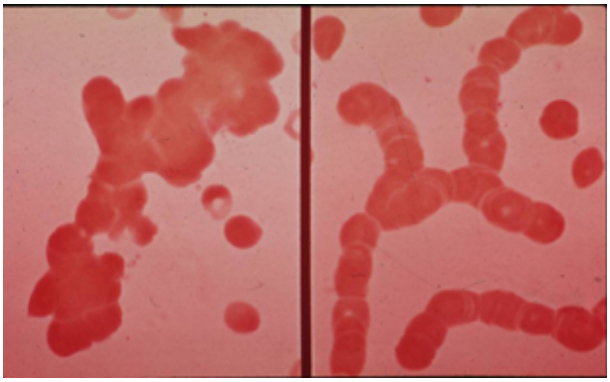

Figure 2.4: Agrupación de Celulas Rojas (izquierda) y formación de Rouleaux (derecha)

puede ser visto como una correlación directa.

Por cuestiones practicas esta relación del estrés cortante en el plasma es normalmente considerado lineal considerando a este como un fluido Newtoniano, sin embargo bajo poca presión el plasma se comporta como un fluido no newtoniano.

2.1.3.1 Viscosidad de la sangre

Una célula roja humana tiene la forma de un disco aplastado en su centro, con un diámetro de 7.2 μm y un ancho promedio de 2.0 μm, y en su parte mas interna tiene un ancho promedio de 1 μm.

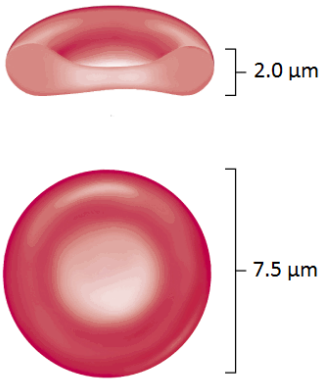

Figure 2.5: Froma biconcava de una célula roja

Un glóbulo rojo tiene un área promedia de 120 μm^2 y un volumen de 85 μm^3, con una forma bicóncava que incrementa el área de difusión de oxígeno, así mismo permite a los glóbulos rojos cambiar su volumen sin romper la membrana que encierra la hemoglobina, esta membrana esta compuesta por una proteína semi-permeable que

permite la entrada de nutrientes y salida de subproductos, esta forma bicóncava de las células rojas no es la única forma que puede tener y dependerá del medio en donde se encuentre para cambiar de forma, por ejemplo si un glóbulo rojo se encuentran en la solución muy saturada de sales, este se hinchara y tendrá una forma mas redonda, por el contrario puede colapsar y parecer un globo desinflado en una solución con pocas sales, como se muestra en la figura 2.6.

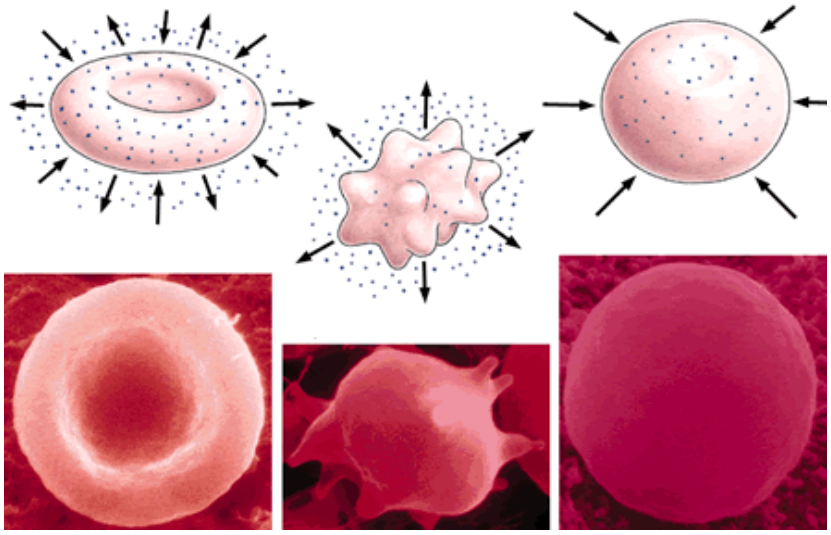

Figure 2.6: Forma bicóncava de una célula roja, izquierda forma bicóncava, centro en una solución hipotónica derecha en una solución hipertónica

Es necesario considerar diferentes parámetros biológicos para aproximar el comportamiento de la sangre de manera realista, uno de estos parámetros que son considerados son los niveles de salinidad en el torrente sanguíneo, este parámetro afecta la forma de las células rojas y por lo tanto la viscosidad de la sangre, también se considera la presión que se ejerce sobre en flujo sanguíneo por parte de las paredes arteriales y del propio flujo, se considera esta presión para estimar la formación de agrupaciones por parte de los glóbulos rojos, que cambiarán de manera local la viscosidad de la sangre, este comportamiento se aproxima considerando la dirección para cada partícula, que dependiendo de donde apunte los glóbulos rojos, sera el comportamiento con sus células vecinas, esta dirección de los glóbulos rojos se mide respecto a una de las caras bicóncavas, que estará girando dependiendo de la presión aplicada y la velocidad con la que viajan las células.

2.2 Problemas relacionados con el Sistema Circulatorio

Algunas de las enfermedades relacionadas con el sistemas circulatorio, pueden causar estrechamiento o bloqueo en los vasos sanguíneos, cortando el suministro de sangre a los tejidos y a los órganos. Uno de los principales problemas en el sistema circulatorio es la formación de coágulos, ya sea en las arterias o en las venas que causan obstrucción del flujo, lo que puede ocasionar diversos problemas perceptibles como son, dolor agudo localizado y cosquilleo en la zona afectada, y de persistir estos bloqueos, estos pueden ocasionar cianosis, que es la decoloración de la piel debido a la pobre circulación en los vasos sanguíneos, así mismo puede provocar perdida de sensibilidad, reflejos y coordinación motriz a largo plazo.

Un embolo es una masa transportada por el flujo sanguíneo, que puede viajar libremente por el torrente sanguíneo hasta llegar a arterias cada vez mas delgadas, donde finalmente obstruyen el flujo hasta formar bloqueos, un embolo puede ser causado por un pedazo de coágulo que se desprende de una problema existente o puede consistir de aire, grasa, células de un tumor u otros materiales. Los émbolos arteriales generalmente son originados en el corazón, lo que significa que estos pueden viajar a cualquier parte del cuerpo, y llegar a órganos como el cerebro, riñones, o bien seguir por los vasos sanguíneos, hasta llegar a las extremidades del cuerpo donde no les es posible continuar obstruyendo el flujo sanguíneo, estos émbolos además de formarse en el corazón también pueden formarse en las bifurcaciones de grandes arterias incluyendo la aorta y la ilíaca, donde depósitos de grasa pueden formarse.

Problema relacionados con el sistema circulatorio también puede encontrarse fuera de los vasos sanguíneos, estos pueden ser obstruidos por fuerzas externas, como ejemplo tenemos la presión que ejerce un torniquete aplicado a una herida, la inflamación debido a enfermedades como la vasculitis, aneurismas que pueden desarrollarse por debilidad en las paredes arteriales, estas causas externas no tienen que ser casos extremos, también pueden ser debidos a acciones cotidianas como es la presión que ejerce el sacro a los vasos sanguíneos al sentarse en una silla, el uso de zapatos ajustados que limitan el flujo sanguíneo a los pies, e inclusive puede presentarse reducción del flujo debido a las posiciones que tomamos al dormir, o bien debido a situaciones que fuera de nuestro control, como por ejemplo corrientes frías que pueden provocar vaso-constricción en vasos arteriales superficiales.

Algunas reducciones en el flujo sanguíneo no siempre son debidos a agentes externos, en algunos casos puede ser originados por la forma en que los vasos se desarrollaron, donde estos crecimientos anormales pueden causar estrechamiento de las arterias, que pueden causar ataques cardiacos, aneurismas, enfermedades vasculares en la periferia, o malformaciones arteriovenosas.

Un problema que debemos enfrentar es el hecho que podemos controlar algunos factores para reducir el riesgo en problemas arteriovenosos, pero no podemos detener el envejecimiento de estas, con la edad una mayor cantidad de depósitos de colesterol

se acumula en las paredes de los vasos sanguíneos, limitando el flujo sanguíneo en estos, o bien los vasos sanguíneos pueden volverse rígidos debido a calcificaciones, estos problemas son presentados con preocupación en personas de 50 años o mas, debido a los hábitos alimenticios. Podemos simplificar en envejecimiento como el cambio en las estructuras de los vasos sanguíneos, los cuales reducen su elasticidad y el área transversal de las arterias, provocando un cambio en el flujo suave y constante de la sangre, con este cambio se elevan los posibles problemas cardiacos que pueden llegar en casos extremos a la muerte.

2.3 Funcionamiento de un Stent

Una de las formas en que se puede aumentar el flujo sanguíneo es mediante el uso de una estructura tubular conocido como stent, este es insertado en la región donde se encuentra parcialmente bloqueada la arteria (arterioesclerosis), este es colocado mediante un catéter que cuenta con un globo donde se ubica el stent, este globo se inflará para colocar de manera permanente la estructura metálica, y una vez colocado, este permitirá restaurar el flujo sanguíneo, sin embargo el uso de este dispositivo no impide que se vuelva a formar un bloqueo arterial volviendo a limitar el flujo, a este problema se le conoce como restenosis.

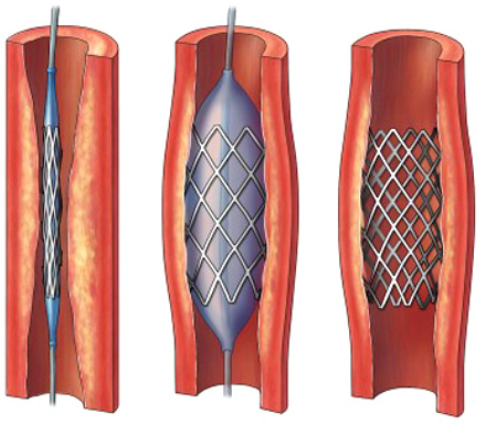

Figure 2.7: Se muestra la colocación de un Stent.

Además del incremento del flujo sanguíneo por parte del stent no se conoce con certeza los posibles problemas que un paciente puede desarrollar a lo largo del tiempo, esto se puede intuir al limitar la flexibilidad de las paredes arteriales que ya no permiten regular la presión y la dirección del flujo sanguíneo al limitar la vasoconstricción y vasodilatación dentro del stent, esto ha sido reportado por varios autores (**? ? ? ?**) mostrando que en zonas cercanas al stent las arterias se comportan de manera inversa

para mantener la presión dentro del segmento que ocupa el stent y en casos extremos ha provocado la muerte en un paciente debido a los medicamentos suministrados (**?**).

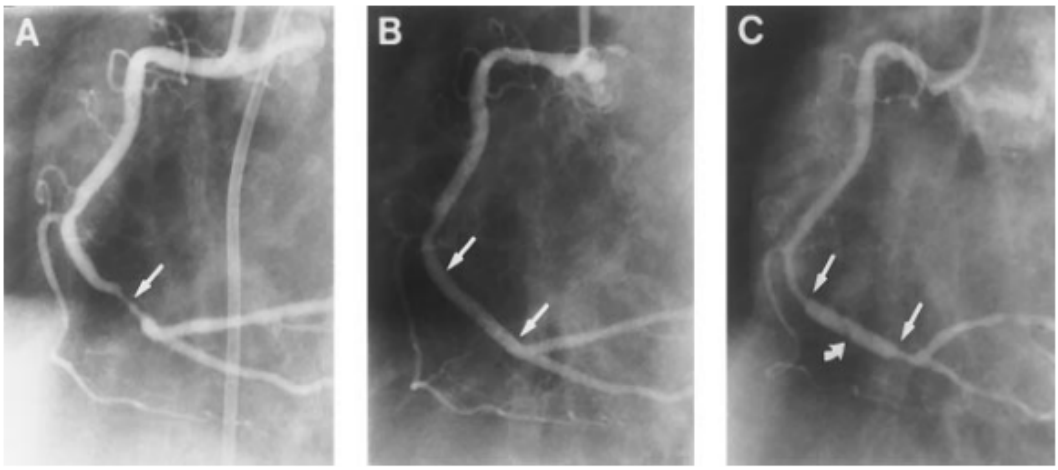

Figure 2.8: A) La arteria muestra una reducción en el flujo sanguíneo, B) Muestra el Stent colocado, C) La arteria donde se coloco el stent muestra vasoconstricción, pero no donde esta colocado el Stent

2.4 Simulación de elementos biológicos

La simulación del fluido biológico de la sangre dependerá en gran medida de entendimiento del problema que se trata de atacar y manejar correctamente los parámetros biológicos, para el trabajo desarrollado se consideraron los siguientes elementos:

- Viscosidad variable de la sangre, ya sea:
  - Global al definir la forma de los glóbulos rojos debido a compuestos en el plasma ó
  - Local al definir la presión la viscosidad de la sangre,
- Expansión y Contracción de los vasos sanguíneos y de manera particular de las arterias elásticas de tamaño medio, este movimiento se considera como respuesta a estímulos en el ambiente,
- Incorporación de objetos ajenos al flujo como es el uso de un catéter, para la colocación de un stent, y
- La incorporación del objeto stent, como un segmento que evita el movimiento de las paredes arteriales que provoca turbulencia al flujo de sangre que pasa por el.

Chapter 3

Mecánica de Medio Continuos

La mecánica de medios continuos puede ser entendido desde dos puntos de vista, desde el punto de vista Euleriana ó por medio de una descripción Lagrangiana. La descripción Euleriana se enfoca en analizar lo que sucede en puntos fijos del espacio mientras transcurre el tiempo de la simulación, mientras que los métodos Lagrangianos sigue la descripción de cada partícula a lo largo de la simulación, esto lo podemos entender con una analogía, imagine una autopista que cuenta con varios carriles, el método Euleriano se enfocaría en observar un punto en específico de algún carril de esta autopista imaginaria y sería el espectador el encargado de estimar cuantos carros pasan en este punto fijo, la velocidad con la que pasan, el modelo de los autos, el color, así como todas las propiedades a estudiar, por otro lado en el método Lagrangiano, el espectador estaría dentro de alguno de estos carros y observaría a su alrededor los automóviles que se mueven a su alrededor mientras avanzan por la autopista, para este ejercicio de imaginación se sugiere que no sea el conductor quien realice las observaciones para evitar colisiones, pero dada la naturaleza caótica de los fluidos tal pareciera que es el conductor quien hace los cálculos mientras conduce.

3.1 Métodos Eulerianos

Los métodos de Volumen Finito y Diferencias Finitas son dos ejemplos de métodos numéricos basados en una descripción Euleriana, hoy en día usados ampliamente para el Cómputo en la Dinámica de Fluidos, sin embargo estos son costosos en términos computacionales cuando se requiere calcular con precisión el movimiento del fluido en fronteras con movimiento, esto es debido a que modificar la malla que define al problema redefiniendola en cada iteración, es dificil y lento, por lo que es difícil agregar objetos que se muevan independientemente al movimiento del flujo simulado, sin mencionar que agregar el estudio de la simulación nuevas propiedades requiere reformular parcial o totalmente el plantemiento de la solución.

3.2 Métodos Lagrangianos

El concepto fundamental detrás de los métodos Lagrangianos o métodos libres de malla es obtener soluciones numéricas a ecuaciones integrales o ecuaciones diferenciales parciales (PDE) usando un conjunto distribuido arbitrario de nodos o partículas sin depender directamente en la conectividad de estos. Los primeros métodos que hacían uso de este planteamiento son los métodos de Marker and Cell (MAC) desarrollado por Harlow y el método de Particle In Cell (PIC) desarrollado Harlow y Welch (21). Estas formulaciones son las primeras en incorporar métodos Lagrangianos para representar el fluido, sin embargo estos no eran completamente Lagrangianos ya que hacian la resolución del movimiento del fluido desde el punto de vista Euleriano. Belytschoko introdujo en 1994 el método de Elemento Libre Galerkin (EFG) (22) que utiliza una interpolación de mínimos cuadrados (MLS), para construir las funciones básicas a resolver y formar las funciones constitutivas, otro ejemplo de un método libre de malla es el método de Partículas de Volumen Finito (FVPM) que fue primeramente introducido por Hietel (23).

Los métodos Lagrangianos al no definir una estructura fija permiten manejar las fronteras del dominio en movimiento, permitiendo interactuar con objetos que no son parte del fluido, pero son dependientes del método utilizado para describir la interacción de las fronteras con el fluido.

3.3 Diferencia entre métodos Eulerianos y métodos Lagrangianos

Otra diferencia marcada entre los métodos Eulerianos y los métodos Lagrangianos es el dominio que estudian, los métodos Eulerianos definen al comienzo de la simulación el dominio de estudio, mientras que dada la naturaleza de los métodos Lagrangianos

son las partículas quienes definen el dominio de estudio, el cual puede expandirse o contraerse dependiendo del movimiento de estas, lo cual es una ventaja cuando se busca una aproximación rápida utilizando pocas partículas a expensas de la precisión, o bien una desventaja si se trata de calcular un gran fluido ya que requiere de una gran cantidad de partículas para representarlo y calcular con exactitud las propiedades deseadas, lo que hace que el manejo de las partículas que representan a este sea más dificil.

3.4 Entendiendo SPH

El método Smooth Particle Hydrodynamics fue originalmente desarrollado por Gingold y Monaghan en 1977 (24), con el objetivo de resolver problemáticas de astrofísica, debido a su facilidad de representar materia en el vacío del espacio, a contrario de los métodos Eulerianos que modelan este vacío como otro fluido de fondo, mientras que el método Lagrangiano reduce los recursos utilizados al solo definir las partículas de estudio; Durante las últimas décadas el método SPH ha ganado madurez y ha sido usado en una gran variedad de problemas de ingenieria, incluyendo pero no limitado a flujos de superficie libre (2, 25), simulación de flujo magmáticos (8), simulación de fluidos sanguíneos (14), y continua siendo una de las mejores herramientas para modelar problemas de astrofísica. El método SPH en esencia toma un conjunto de puntos muestra que son utilizados para describir el fluido, cada uno de estos puntos contienen definiciones físicas del mismo, como la masa, posición, velocidad, entre otras, y al mismo tiempo estas muestras pueden ser usadas para contener otro tipo de entidades físicas, dependiendo del problema que se este resolviendo, y haciendo uso de estas propiedades pueden aproximarse definiciones macroscópicas que son obtenidas por la ponderación que cada partícula proveé a la cantidad física buscada.

Comparado con otros métodos numéricos que de igual forma aproximan las derivadas como lo hace el método de diferencias finita, el método SPH no requiere que las partículas sean alineadas de manera regular en un mallado, el método SPH permite la aproximación de las derivadas de un campo continuo permitiendo la ubicación de las partículas de manera casi arbitraria, debido a que cada una de estas muestras ocupa un punto en el espacio y es parte del problema, y para obtener un resultado más preciso se requiere que el conjunto de partículas sea mayor, así mismo la función de interpolación juega un importante rol en el método SPH, ya que este permitirá o no una distribución mas o menos aleatoria para estimar la propiedad física deseada de manera correcta.

La descripción del método SPH propuesto por Gingold y Monaghan (24), describen los dos principales fundamentos para comprender el método SPH: 1) la función de aproximación y 2) el kernel de suavizado, entenderlos es fundamental para entender las limitaciones y ventajas del uso del método SPH.

3.4.1 Función de Aproximación

El concepto detrás de la función de aproximación, de una función $f(\bar{x})$ usada en SPH proviene de la identidad:

$$f(\bar{x}) = \int_{\Omega} f(\bar{x}')\delta(\bar{x} - \bar{x}')\mathrm{d}\bar{x} \tag{3.1}$$

Donde f es una función determinada por el vector de posición x, definida en un dominio Ω y $\delta(\bar{x} - \bar{x}')$ es la función delta de Dirac que está definida como:

$$\delta(\bar{x} - \bar{x}') = \begin{cases} +\infty & \bar{x} = \bar{x}' \\ 0 & \bar{x} \neq \bar{x}' \end{cases} \tag{3.2}$$

Si la función delta de Dirac es reemplazada por una función arbitraria W, la función de aproximación de $f(\bar{x})$, está dado por:

$$\tilde{f}(\bar{x}) = \int_{\Omega} f(\bar{x}')W(\bar{x} - \bar{x}', \mathrm{h})\mathrm{d}\bar{x} \tag{3.3}$$

Donde h es conocido como el parámetro de suavizado, el cual determina el dominio de acción de la función kernel.

3.4.2 Kernel de Aproximación

El uso de diferentes kernels en SPH es análogo a usar diferentes esquemas para el método de diferencia finita, es importante hacer incapie en la importancia de elegir adecuadamente una función kernel dependiendo de la propiedad física estudiada, sin embargo las principales características de cualquier función W debe tener, son expuestas por J.J. Monahan (9) las cuales pueden ser estimadas de la identidad 3.1, estas dos propiedades fundamentales que cualquier kernel W deben cumplir son:

$$\int_{\Omega} W(\bar{r}, \mathrm{h})\mathrm{d}\bar{x} = 1 \tag{3.4}$$

$$\lim_{h \to 0} W(\bar{r}, \mathrm{h})\mathrm{d}\bar{x} = \delta(\bar{r}) \tag{3.5}$$

La ecuación 3.4 establece que el kernel debe ser normalizado con el propósito de asegurar que los máximos y mínimos de la función W no sean realzados. El kernel así mismo debe ser positivo para asegurar que el kernel provea una función de aproximación adecuada, así mismo si la función es simétrica, es decir:

$$W(\bar{r}, \mathrm{h}) = W(-\bar{r}, \mathrm{h}) \tag{3.6}$$

La función simétrica W asegura invariación en sistemas rotacionales, por lo que si se elige un kernel que cumpla con la propiedad 3.6 y 3.4, es decir que el kernel sea simétrico y normalizado, el error de aproximación para la propiedad interpolada será de

segundo orden. Las funciones kernel mas simples de elegir utilizan una función Gaussiana acotadas en un rango, debido a su simplicidad y que cumplen con las propiedades de ser simétrica, positiva, compacta y satisfacer la condición unitaria al establecer la constante adecuada dependiente de la función gaussiana elegida 3.4.

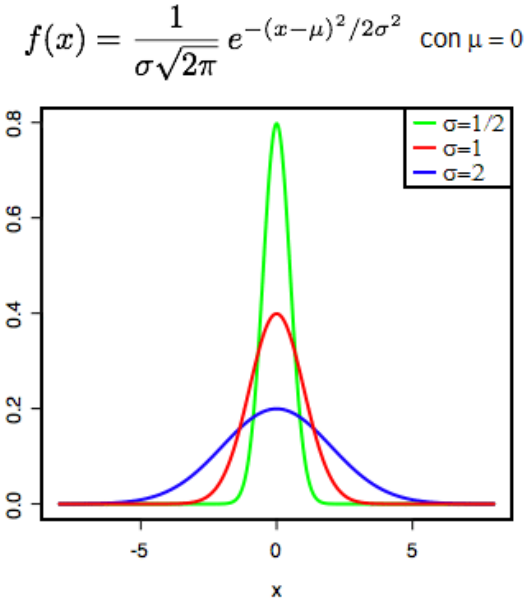

Figure 3.1: Función Gaussiana con diferentes valores σ.

Trabajos recientes explican que la utilización de funciones Gaussianas no es la mejor elección como Kernel de Aproximación, como lo demuestra F. M. Lang, A. S. Iglesias y M. A. and A. Colagrossi (26), asi como S.P. Korzilius, W.H.A. Schilders, M.J.H. Anthonissen (27) este último trabajo propone el uso de funciones Wendland ya que provee una mejor aproximación sin depender de la posición de las partículas.

Resumiendo, la función Kernel W a ser usada en la Función de aproximación que permita interpolar correctamente la propiedad calculada 3.3, debe cumplir con las siguientes características:

1. $W(\bar{x} - \bar{x}', \mathrm{h}) = W(\bar{x}' - \bar{x}, \mathrm{h}),$
2. $W(\bar{x} - \bar{x}', \mathrm{h}) \geq 0,$
3. $W(\bar{x} - \bar{x}', \mathrm{h}) = 0, \text{para } |\bar{x} - \bar{x}'| \geq \mathrm{h}$
4. $\int_{\Omega} W(\bar{x} - \bar{x}', \mathrm{h}) \mathrm{d}\bar{x} = 1,$
5. $\lim_{h \to 0} W(\bar{x} - \bar{x}', \mathrm{h}) = \delta(\bar{x} - \bar{x}')$

Donde k, es una constante que define el dominio donde la función kernel esta definida, este dominio, $|\bar{x} - \bar{x}'| \leq \mathrm{h}$, es referido como el dominio de soporte, para una posición $\bar{x}$ arbitraria.

El equivalente discreto de la función $\tilde{f}(\bar{x})$ en la ecuación 3.3 es calculado mediante la aproximación de la integral como una sumatoria de los elementos próximos al punto que se están evaluando:

$$A_s(\bar{x}) = \sum_{j=1}^{N} A_j V_j W(\bar{x} - \bar{x}_j, \mathrm{h}) \tag{3.7}$$

Cada una de las N partículas es evaluada, donde para cada una de estas se considera el volumen V_j que ocupan en la posición del espacio $\bar{x}_j$, que agrega un aporte A_j en función del kernel W determinando por la distancia de la partícula j en el espacio $\bar{x}_j$ y el punto evaluado $\bar{x}$ proporcional al valor de la propiedad A_s evaluada.

3.4.3 Error del Kernel de Aproximación

Asumiendo que $f(\bar{x})$ es suficientemente suave y diferenciable, el error de truncamiento en el kernel de aproximación puede ser estimado utilizando series de Taylor para $f(\bar{x}')$ cerca de $\bar{x}$. Sustituyendo la expansión en la ecuación 3.3 y considerando el dominio de soporte de la función kernel $|\bar{x} - \bar{x}'| \geq \mathrm{h}$, podemos expresar la ecuación como:

$$\begin{aligned}
\tilde{f}(\bar{x}) &= \int_{\Omega} [f(\bar{x}) + f'(\bar{x}' - \bar{x}) + \mathcal{O}((\bar{x} - \bar{x}')^2)] W(\bar{x} - \bar{x}', \mathrm{h}) \mathrm{d}\bar{x} \\
&= f(\bar{x}) \int_{\Omega} W(\bar{x} - \bar{x}', \mathrm{h}) \mathrm{d}\bar{x} \\
&+ \int_{\Omega} f'(\bar{x}' - \bar{x}) W(\bar{x} - \bar{x}', \mathrm{h}) \mathrm{d}\bar{x} + \ \mathcal{O}(\mathrm{h}^2)
\end{aligned} \tag{3.8}$$

Debido a que W es una función simétrica, podemos hacer reducir a:

$$\int_{\Omega} f(\bar{x}' - \bar{x}) W(\bar{x} - \bar{x}', \mathrm{h}) \mathrm{d}\bar{x} = 0 \tag{3.9}$$

Usando el resultado anterior y sustituyendo en 3.8 obtenemos:

$$\tilde{f}(\bar{x}) = f(\bar{x}) + \mathcal{O}(\mathrm{h}^2) \tag{3.10}$$

De lo anterior podemos ver que el kernel de aproximación de una función, tiene un orden de error de segundo grado, siempre y cuando la función kernel sea una función par, y cumpla la condición de unidad.

3.4.4 Discretización del dominio de trabajo

La aproximación de una propiedad A_s requiere la evaluación de las partículas que se encuentran dentro del dominio de integración de la partícula i, como se muestra en la figura 3.2.

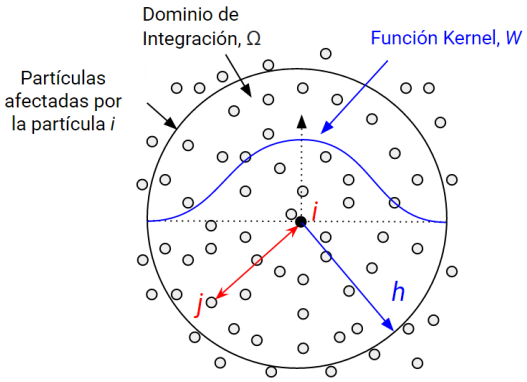

Figure 3.2: Particula i evaluada, y el conjunto de partículas afectadas por esta.

Donde para una partícula, i, la aproximación de la función A_s pueden ser obtenida por la aproximación 3.7 al evaluar solo las partículas que están dentro del dominio definido por la función kernel W:

$$\tilde{f}(\bar{x}_i) = \sum_{j=1}^{N} f(\bar{x}_j) W(\bar{x}_{ij}, \mathrm{h}) \Delta V_j \tag{3.11}$$

$$\nabla \cdot \tilde{f}(\bar{x}_i) = \sum_{j=1}^{N} f(\bar{x}_j) \nabla_i W(\bar{x}_{ij}, \mathrm{h}) \Delta V_j \tag{3.12}$$

Donde:

$\bar{x}_{ij}$	Es una simplificación de $\bar{x}_i - \bar{x}_j$
N	Es el número de partículas del sistema
W	Es la función kernel
ΔV_j	Es el volumen que ocupa la partícula j
$\nabla_i W$	Es el gradiente de la función kernel respecto a la partícula i

El volumen de cada partícula es determinado por la identidad:

$$m_j = \Delta V_j \rho_j \tag{3.13}$$

Donde ρ_j es la densidad de la partícula j, sustituyendo en las ecuaciones 3.11 y 3.12 con esta identidad obtenemos:

$$\tilde{f}(\bar{x}_i) = \sum_{j=1}^{N} \left(\frac{m_j}{\rho_j} \right) f(\bar{x}_j) W(\bar{x}_{ij}, \mathrm{h}) \tag{3.14}$$

$$\nabla \cdot \tilde{f}(\bar{x}_i) = \sum_{j=1}^{N} \left(\frac{m_j}{\rho_j} \right) \nabla_i f(\bar{x}_j) W(\bar{x}_{ij}, \mathrm{h}) \tag{3.15}$$

Es decir, la aproximación de una función escalar así como sus derivadas espaciales para un punto $\bar{x}_i$, puede ser aproximadas por la aportación de todas las partículas que están dentro del dominio, que define la función kernel W para la posición $\bar{x}_i$.

En la práctica la aproximación del gradiente de una función (3.15) tiende a ser inexacta e inestable, debido a que la interacción entre las partículas no necesariamente es simétrica, es por ello que dos alternativas son presentadas por Monaghan, las cuales proveen una mayor precisión y estabilidad dentro de la simulación, estas propuestas son:

$$\nabla \cdot f(\bar{x}) = \frac{1}{\rho} \Big[\nabla \cdot \big(\rho f(\bar{x}) \big) - f(\bar{x}) \cdot \nabla \rho \Big] \tag{3.16}$$

$$\nabla \cdot f(\bar{x}) = \rho \Big[\nabla \cdot \Big(\frac{f(\bar{x})}{\rho} \Big) + \frac{f(\bar{x})}{\rho^2} \cdot \nabla \rho \Big] \tag{3.17}$$

Sustituyendo estas propuestas en la ecuación 3.7, y siguiendo el mismo procedimiento para obtener la aproximación de la partícula, se obtienen las siguientes divergencias de aproximaciones para una propiedad continua $f(\bar{x})$:

$$\nabla \cdot \tilde{f}(\bar{x}_i) = \frac{1}{\rho_i} \left[\sum_{j=1}^{N} m_j \big(f(\bar{x}_j) \big) - f(\bar{x}_i) \cdot \nabla_i W_{ij} \right] \tag{3.18}$$

$$\nabla \cdot \tilde{f}(\bar{x}_i) = \rho_i \left[\sum_{j=1}^{N} m_j \Big(\frac{f(\bar{x}_j)}{\rho_j^2} + \frac{f(\bar{x}_i)}{\rho_i^2} \Big) \cdot \nabla_i W_{ij} \right] \tag{3.19}$$

Donde $W_{ij} = W(\bar{x}_{ij}, \mathrm{h})$ es usado por simplificación, cabe mencionar que las ecuaciones 3.18 y 3.19 presentan propiedades diferentes a la ecuación 3.15. En la ecuación 3.18 podemos apreciar que es antisimétrica, y generalmente es usada para reproducir el campo físico del gradiente de velocidad en la ecuación de continuidad, mientras que la simetría de la ecuación 3.19 es usada para el uso de la discretización del gradiente de presión en la ecuación de momento, al asegurar que el momento del sistema sea conservativo.

Es importante entender que el método SPH por sí mismo acarrea algunas problemáticas inherentes, debido a la discretización empleadas en el método, así como el uso de la función kernel, un ejemplo de esto es cuando se usa SPH para derivar el movimiento de las partículas, la forma en que se realiza no satisfacen ciertos principios

físicos como la simetría de fuerzas y la conservación de momento, sin embargo se tratan de compensar estas deficiencias con el uso de diferentes kernels W para aproximar las funciones deseadas, es por ello que dependerá de la propiedad física el kernel usado, ejemplo de esto son las ecuaciones 3.18 y 3.19.

Es importante determinar la función kernel que será utilizado para determinar la aportación de cada una de las partículas vecinas, como lo demuestra Walter Dehnen y Hossam Aly (28) , es por ello que se escogió una función kernel Wendland, de grado 3 debido a que presenta todas las características deseadas en una función kernel (positivo, simétrico, unitario, compacto) y ha demostrado un mejor desempeño que el uso de funciones b-spline de alto orden:

$$W_{3,3} = \frac{\sigma_W}{h^3}(1-q)_+^8(32q^3 + 25q^2 + 8q + 1) \tag{3.20}$$

Donde:

$(.)_+$ Es una función que simplifica la función $max(., 0)$

q Es la normalización de la distancia y el parámetro de suavizado de la función kernel. Donde $q = \dfrac{|\bar{x} - \bar{x}'|}{\mathrm{h}}$

σ_W Es el valor de normalización de la función kernel estos valores son: $\dfrac{78}{7\pi}$ y $\dfrac{1365}{64\pi}$ para 2 y 3 dimensiones respectivamente.

3.5 Ecuaciones Gobernantes

Las principales propiedades físicas de un fluido isotérmico y viscoso considerados en esta tesis son la velocidad $\bar{u}$, masa-densidad ρ y presión $\bar{p}$, las cuales son interpoladas utilizando las respectivas funciones kernel para cada propiedad, con el propósito de estimar los valores de los campos continuos a lo largo del fluido. La formulación clásica para el movimiento del fluido a lo largo de un tiempo t esta dado por las ecuaciones de Navier-Stokes:

$$\rho\Big(\frac{\delta}{\delta t} + \bar{u}\cdot\nabla\Big)\bar{u} = -\nabla\bar{p} + \mu\nabla\cdot(\nabla\bar{u}) + f \tag{3.21}$$

Donde para un punto evaluado:

μ Es la viscosidad del fluido

$\bar{u}$ Es el vector de velocidad del fluido

f Son las fuerzas externas que actúan sobre el fluido

ρ Es la masa-densidad del fluido

$\bar{p}$ Es la presión del fluido

Sin embargo la descripción dada por la ecuación 3.21 define el flujo para una descripción Euleriana, al representar un fluido cualquiera en un esquema Lagrangiano, las ecuaciones se simplifican un poco, una de las razones es que se asume que la cantidad de partículas es constante a lo largo de la simulación, por lo que el mantener la masa fija en cada una de las partículas, la ley de conservación de masa se satisface sin necesidad de realizar algún cálculo extra, pero también acarrea algunas problemáticas como la incompresibilidad de un flujo ya que no es posible simplemente definir el gradiente de velocidad igual a cero para lograrlo.

$$\nabla \cdot \bar{u} = 0 \tag{3.22}$$

Así mismo en la descripción Lagrangiana de un fluido, las partículas definen completamente a este, lo que significa que las partículas se mueven con el fluido, lo que comparado con la descripción Euleriana donde cualquier valor que se quiera estudiar depende de un tiempo t, los método Lagrangianos en general y de manera particular para el método empleado de SPH, las propiedades como la masa, velocidad y posición están representadas por las partículas que describen el fluido, lo que simplifica el calculo de algunas propiedades como la aceleración, al solo ser necesario derivar respecto al tiempo t, la velocidad de la partícula para obtener esta propiedad, esta representación implica la eliminación del termino de advección de la ecuación 3.21, por lo que para una descripción Lagrangiana las ecuaciones de Navier-Stoles para un fluido viscoso e isotérmico esta dado por:

$$\rho \frac{d\bar{u}}{dt} = -\nabla \bar{p} + \mu \nabla^2 \bar{u} + \mathrm{f} \tag{3.23}$$

Donde los términos de lado derecho definen las fuerzas internas y externas que actúan para sobre la partícula, ambos campos de fuerza pueden ser combinado en:

$$\mathrm{F} = \mathrm{f}_{interna} + \mathrm{f}_{externa}$$

Donde:

$$\mathrm{f}_{interna} = -\nabla \bar{p} + \mu \nabla^2 \bar{u}$$

Que representa la presión y la viscosidad del fluido. Así mismo el determinar la aceleración de un partícula i, esta dada por:

$$a_i = \frac{d\bar{u}_i}{dt} = \frac{\mathrm{F}_i}{\rho_i} \tag{3.24}$$

Donde $\bar{a}_i$ y $\bar{u}_i$ son la aceleración y la velocidad de la partícula i respectivamente y F_i es la fuerza total que actúa sobre la partícula ρ_i que es la masa-densidad evaluada en la posición de la partícula i.

3.5.1 Masa-Densidad

Cualquier propiedad física para una posición dada, así como su gradiente o su derivada espacial de gradiente puede ser aproximado usando la ecuaciones 3.11 y 3.12 respectivamente, para evaluar la aportación que cada una de las partículas proporciona en la posición evaluada, en este caso en particular la densidad. Dada la naturaleza del método de SPH, la conservación de masa para todo el sistema se conserva, pero no así la densidad en cada uno de los puntos del espacio, por lo que el calcular el campo físico de densidad debemos sustituir en la ecuación 3.11 la función escalar que queremos calcular, obteniendo:

$$\begin{aligned} \rho_i &= \rho(\bar{x}_i) \\ &= \sum_j \rho_i \frac{m_j}{\rho_j} W(\bar{x}_i - \bar{x}_j, \mathrm{h}) \end{aligned} \tag{3.25}$$

$$= \sum_j m_j W(\bar{x}_i - \bar{x}_j, \mathrm{h}) \tag{3.26}$$

Donde:

$\bar{x}_i$	Es la posición de la partícula i
ρ_j	Es la densidad de la partícula j
m_j	Es la masa de la partícula j
W	Es la función kernel

Es posible simplificar la ecuación 3.25 si se considera que todas las partículas contienen la misma masa y ocupan el mismo volumen, lo que da lugar a la ecuación 3.26.

3.5.2 Fuerzas Internas

Las fuerzas internas son debidas a la interacción del fluido consigo mismo, ejemplo de estas fuerzas con la presión y la viscosidad representadas en los dos primeros términos de la derecha en la ecuación 3.23.

3.5.2.1 Presión

La presión p para una determinada partícula puede ser aproximada utilizando la ley de los gases ideales:

$$\bar{p}V = nRT \tag{3.27}$$

Donde:

V Es el volumen por unidad de masa, que es determinado como $V = \frac{1}{\rho}$

n Es el número de partículas del gas por mol.

R Es la constante universal de los gases.

T Es la temperatura medida en Kelvins.

Para un fluido isotérmico con masa constante, podemos sustituir de la ecuación 3.27 la parte de derecha de por una constante k, que teóricamente solo depende de la cantidad de partículas que contiene el fluido, lo que se simplifica a:

$$\begin{aligned} pV &= k \\ p\frac{1}{\rho} &= k \\ p &= k\rho \end{aligned} \tag{3.28}$$

Si la presión de cada partícula i es conocida la fuerza interna debida a la presión, puede ser calculada como:

$$\begin{aligned} \mathrm{f}_i^{\text{presión}} &= -\nabla p(\mathrm{r}_i) \\ &= -\sum_{j \neq i} p_j \frac{m_j}{\rho_j} \nabla W(r_i - r_j, \mathrm{h}) \end{aligned} \tag{3.29}$$

Desafortunadamente la presión aplicada para cada partícula usando la ecuación anterior no es simétrica, esto puede ser verificado en la interacción de dos partículas que usa la presión de la partícula que esta evaluando para calcular su fuerza de presión y viceversa. Debido a esto la fuerza calculada con la ecuación 3.29, sera incorrecta, una manera de solucionar este problema es mediante el uso de la ecuación 3.19 y siguiendo el mismo procedimiento realizado con 3.29, la ecuación que nos describe la presión correctamente esta dada por:

$$\mathrm{f}_i^{\text{presión}} = -\rho_i \sum_{j \neq i} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W(\mathrm{r}_i - \mathrm{r}_j, \mathrm{h}) \tag{3.30}$$

Otra posible solución que hace referencia M. Müller y M.Gross (29), es mediante:

$$\mathrm{f}_i^{\text{presión}} = -\rho_i \sum_{j \neq i} \frac{p_i + p_j}{2} \frac{m_j}{\rho_j} \nabla W(\mathrm{r}_i - \mathrm{r}_j, \mathrm{h}) \tag{3.31}$$

Ambas ecuaciones permiten calcular la fuerza de presión de manera simétrica, lo que permite que las propiedades de momento lineal y angular sean conservativas, sin embargo la última ecuación requiere un menor tiempo computacional. El uso de la ecuación de los gases ideales funciona como se espera en un entorno de vacío donde

los gases tienden a estar siempre en repulsión y en constantes expansión hasta ocupar el volumen del contenedor que los contiene, sin embargo no queremos este tipo de comportamiento en los fluidos, lo cuales alcanzan un punto de equilibrio cuando la fuerza de presión entra en un estado de reposo.

La propuesta que varios autores hacen referencia es la adecuación a la ecuación 3.29 al agregar el término de p_0 referente a la presión en reposo, esta adecuación resulta en:

$$\begin{aligned}(p + p_0)V &= k \\ p + kp_0 &= k\rho \\ p &= k(\rho - \rho_0)\end{aligned} \tag{3.32}$$

Con este término podemos tener un fuerza de presión que presenta una acción de repulsión/atracción hasta que se alcanza un estado de equilibrio para simular la atmósfera que ejerce sobre cualquier fluido, considerando que el flujo es isotérmico y con masa constante, de lo contrario la constante k debe ser cambiada por una función dependiente de la posición de la partícula.

Es necesario señalar que el limite que se ponga como presión de reposo, debe ser acorde al kernel elegido para estimar la fuerza de presión, debido a que la fuerza de repulsión debe incrementarse mientras mas cerca se estén de las partículas, y debe atenuarse conforme se llegue al punto de equilibro, es por ello que se eligió un kernel similar al propuesto por Desbrun (30) el cual propone una solución que evita la formación de cúmulos utilizando un kernel menos suavizado, la propuesta de kernel para el cálculo de la presión es:

$$W_{\text{presión}}(\mathrm{r}, \mathrm{h}) = \begin{cases} (\mathrm{h} - \|\mathrm{r}\|)^3 & 0 \leq \|\mathrm{r}\| \leq \mathrm{h} \\ 0 & \|\mathrm{r}\| > \mathrm{h} \end{cases} \tag{3.33}$$

Donde el gradiente del kernel usado es:

$$\nabla W_{\text{presión}}(\mathrm{r}, \mathrm{h}) = -\frac{45}{\pi \mathrm{h}^6} \frac{\mathrm{r}}{\|\mathrm{r}\|} (\mathrm{h} - \|\mathrm{r}\|)^2, \tag{3.34}$$

$$\lim_{\mathrm{r} \to 0^-} \nabla W_{\text{presión}}(\mathrm{r}, \mathrm{h}) = \frac{45}{\pi \mathrm{h}^6}, \quad \lim_{\mathrm{r} \to 0^+} \nabla W_{\text{presión}}(\mathrm{r}, \mathrm{h}) = -\frac{45}{\pi \mathrm{h}^6}$$

Donde el laplaciano del kernel usado es:

$$\nabla^2 W_{\text{presión}}(\mathrm{r}, \mathrm{h}) = -\frac{90}{\pi \mathrm{h}^6} \frac{\mathrm{r}}{\|\mathrm{r}\|} (\mathrm{h} - \|\mathrm{r}\|)(h - 2\|\mathrm{r}\|) \tag{3.35}$$

$$\lim_{\mathrm{r}to0} \nabla^2 W_{\text{presión}}(\mathrm{r}, \mathrm{h}) = -\infty$$

3.5.2.2 Viscosidad

El cálculo de la fuerza interna debido a la viscosidad es estimada con la siguiente ecuación:

$$\begin{aligned} \mathrm{f}_i^{\text{viscosidad}} &= \mu \nabla^2 \bar{u}(\bar{x}_i) \\ &= \mu \sum_{j \neq i} \bar{u}_j \frac{m_j}{\rho_j} \nabla^2 W(\bar{x}_i - \bar{x}_j, \mathrm{h}) \end{aligned}$$

Sin embargo esta ecuación no presenta simetría en las fuerzas calculadas entre las partículas y agrega una fuerza inexistente, por lo que Müller propone una ecuación alternativa:

$$\mathrm{f}_i^{\text{viscosidad}} = \mu \sum_{j \neq i} (\bar{u}_j - \bar{u}_i) \frac{m_j}{\rho_j} \nabla^2 W(\bar{x}_i - \bar{x}_j, \mathrm{h}) \tag{3.36}$$

Esta ecuación hace uso de la velocidad relativa entre ambas partículas para estimar la fuerza de fricción entre estas, de esta manera si dos partículas que interactúan con la misma velocidad no agregará una fuerza fantasma adicional.

3.5.3 Fuerzas Externas

Las fuerzas externas son balanceadas con las fuerzas internas, como se muestra en la ecuación 3.21 donde se indica todas las fuerzas externas como un solo término f, que es la suma de todas la fuerzas externas que actúan sobre la partícula evaluada, renombrando este termino tenemos:

$$\mathrm{f} = \mathrm{f}^{\text{externas}} = \sum \mathrm{f}^l \tag{3.37}$$

Donde l, indica el número de fuerzas externas que son aplicadas en la simulación. Algunas fuerzas pueden ser aplicadas sin la necesidad de revisar las partículas vecinas, como es el caso de la gravedad, (para el caso particular de la simulación del flujo sanguíneo), o si se está modelando un campo eléctrico sólo se necesitará de la posición de la partícula eléctricamente cargada para saber la la dirección y magnitud de la fuerza que debe ser aplicada, otras fuerzas requerirán de aplicar el kernel de aproximación a las partículas vecinas para calcular la dirección y magnitud de la fuerza, como es el caso de la tensión superficial.

3.5.3.1 Gravedad

La fuerza gravitacional es aplicada de igual manera a cada una de las partículas del fluido, y está dada por:

$$\mathrm{f}_i^{\text{gravedad}} = \rho_i \bar{g} \tag{3.38}$$

Donde $\bar{g}$ es el vector que expresa la dirección y la magnitud de la aceleración gravitacional.

3.5.3.2 Flotabilidad

En general para cualquier fluido requerimos que las partículas tengan flotabilidad, La flotabilidad es causada por la difusión de las temperaturas, para un fluido isotérmico, podemos estimar una flotabilidad artificial dado la densidad de la partícula, que podemos expresar como:

$$\mathrm{f}_i^{\,\mathrm{flotabilidad}} = \mathrm{b}(\rho_i - \rho_0)\bar{g} \tag{3.39}$$

Donde $b > 0$ es el coeficiente de flotabilidad artificial, la fuerza de flotabilidad causará que las partículas de menor masa-densidad sigan una trayectoria contraria definida por el vector de gravedad $\bar{g}$, y permitirá que los objetos con mayor densidad se muevan en favor del vector de gravedad.

3.6 Ventajas y Limitantes

Debido a la naturaleza Lagrangiana del método SPH, este provee algunas ventajas comparadas con el uso de métodos Eulerianos:

- El método SPH fue planteado originalmente para la simulación de problemas astrofísicos, con el propósito de solo simular aquellas regiones de interés, lo que dirige los costos computacionales solo en las regiones que existen partículas, a comparación de los métodos eulerianos que tratan en vacío del espacio como un fluido de fondo ocupando recursos computacionales para representarlo.
- No existen restricciones geométricas en el inicio o el desarrollo de la simulación ya que no está restringido a un complicado mallado, lo que permite el cambio de las condiciones de frontera de manera sencilla.
- Agregar una nueva ecuación gobernante es de manera directa en el código desarrollado.
- Cada partícula define el dominio de estudio, lo que permite la expansión o contracción del espacio dependiendo de las ubicaciones de las partículas.

Sin embargo el método no es perfecto y cuenta con sus propias limitantes, como son:

- Aunque se han desarrollado nuevas técnicas para manejar las condiciones de frontera aun se tiene el problema de la penetrabilidad de las partículas dentro de los cuerpos que definen estas fronteras.

- La exactitud del método está definido por el número de muestras (partículas) que se tengan del fluido así como de la distribución inicial de estas, si las muestras no se toman homogéneamente se podría estar definiendo zonas más densas en algunas partes, y aunque a él mismo método se autocorrige en la posición de las partículas, la posición inicial de las partículas puede dar lugar a problemas en el resultado si el problema es dependiente del estado inicial.

- El cálculo de las vecindades de las partículas puede llevar a un gran coste computacional si no se realiza de manera adecuada, lo que en comparación con los métodos malladas, puede ser más lento.

Es necesario mencionar que las condiciones de frontera en particular, las fronteras sólidas permanece como un reto en el método SPH, Monahan (1992) (9) propuso una solución de fuerza de repulsión tipo Lennard-Jones para prevenir que el fluido penetrara en la frontera de un sólido arbitrario, este tipo de solución aunque fácil de implementar, provoca inestabilidad cerca de las fronteras lo que limita el delta de simulación cerca de la frontera de los sólidos. Otro tipo de solución propuesta por Colagrossi y Landrini (6), hace uso de la partículas fantasma, al colocar en la frontera de los sólidos, partículas que transmiten la fuerza a la frontera del sólido y son procesadas similarmente a las partículas del fluido estudiado, excepto que estas partículas fantasma no se mueven libremente y están sujetas al movimiento del sólido, esta aproximación muestra mayor estabilidad cerca de las fronteras del sólido, sin embargo está limitado por la complejidad de la geometría del sólido, ya que requiere calcular la posición inicial de las partículas fantasma al inicio de la simulación, o cuando la frontera del sólido se deforma, otra solución propuesta por Dalrymple y Knio (31) resuelve esta problemática haciendo uso de capas de partículas alrededor del sólido, la posición de estas partículas es previamente calculando alrededor de los objetos sólidos y de las fronteras, a esta solución también se le conoce como Dynamic Boundary Particle (DBP), esta solución aproxima la interacción con las fronteras del sólido y es la mejor propuesta que ha presentado mejores resultados respecto al uso de partículas fantasma, sin embargo no garantiza que el fluido no sea penetrado por el fluido, así mismo si el objeto es sometido a grandes deformaciones es necesario volver a calcular las capas de las partículas, para evitar la penetración de las partículas del fluido al objeto.

El presente trabajo hace uso de un sistema híbrido que incorpora una sola capa de partículas sobre la frontera del sólido, en este caso las arterias las cuales ejercen una fuerza de repulsión para que las partículas no penetren el objeto, mientras que partículas fijas ubicadas en el exterior de la frontera proveen la información necesaria para simular la dilatación y contracción de las paredes arteriales.

Chapter 4

Implementando SPH

Una de las principales problemáticas que involucra la implementación del método SPH, es la gestión de los recursos involucrados ya sea por parte de las partículas que representan el fluido, la definición de fronteras del fluido que puede o no tener una geometría sencilla, y en particular para esta implementación la distribución de recursos entre distintas máquinas ya que involucra una serie de problemáticas como lo es la perdidas de paquetes, la desincronización de uno o varios nodos y en el peor de los escenarios la desconexión de uno o varios nodos.

El propósito de este capítulo es presentar las propuestas a las problemáticas antes descritas, presentando los posibles escenarios donde se presenta un mejor y peor rendimiento respecto a la solución presentada, justificando los recursos involucrados, así como los limites para evitar los escenarios de peor desempeño.

4.1 Partículas Vecinas

La forma más simple y poco óptima de saber si una partícula está dentro del rango de otra, es haciendo una búsqueda exhaustiva de esta contra todas las demás, por lo que para evaluar una sola partícula tenemos una complejidad $\mathcal{O}(n)$, y para evaluar todas las partículas nos tomaría $\mathcal{O}(n^2)$ comparaciones lo que hace este método extremadamente lento, lo que se propone es reducir el espacio de búsqueda organizando las partículas en sectores.

4.1.1 Concepto de Búsqueda en una recta

Para entender el concepto de sectores nos apoyaremos de una analogía, imagine una linea recta y sobre ésta existen partículas dispersas al azar, se desea para cada partícula buscar todas las partículas que están a lo más L de distancia una de otra, una forma de lograr esto es dividir para cada partícula su posición en la recta por un valor L, truncar el valor entero y almacenarlo, de esta forma todas las partículas con el mismo valor se encontrarán a lo mas L de distancia entre sí, por lo que no hay necesidad de

hacer comparaciones de una partícula contra otra del mismo sector, solo será necesario comparar esta partícula contra todas las partículas de los sectores contiguos, lo que reduce de manera significativa el espacio de búsqueda, como se muestra en la figura 4.1.

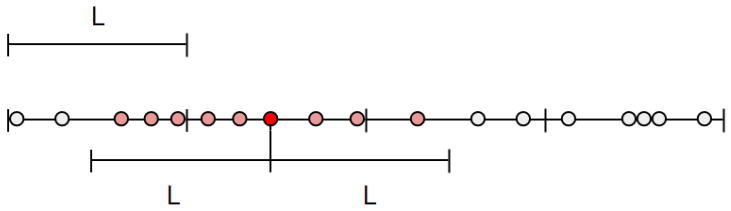

Figure 4.1: Representación de la búsqueda de las partículas en una recta haciendo uso de sectores.

Este tipo de solución será óptimo si existen cúmulos de partículas separados entre sí, ya que los sectores donde se encontraran estas partículas tendrán pocos o ningún sectores contiguos, esta propuesta permite reducir el número de partículas a ser evaluadas de manera general, e inclusive es posible evitar la búsqueda de partículas si los cúmulos de partículas se encuentran encerrados en sectores únicos, ya que todas las partículas se encontraran a lo más a L de distancia entre sí, por lo que no habrá necesidad de evaluar ninguna partícula.

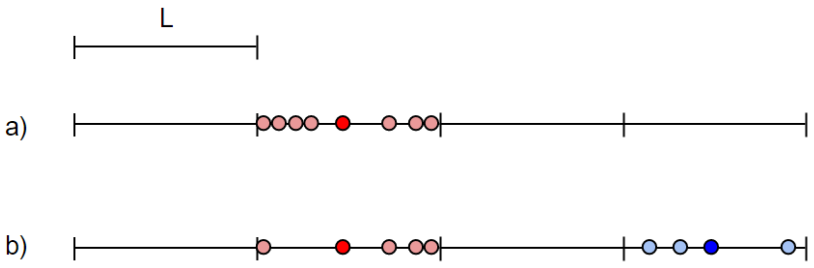

Figure 4.2: Casos óptimos haciendo uso de sectores en una recta: a) todas las partículas en un solo sector, y b) cúmulos de partículas definidos, encerrados en sectores únicos.

4.1.2 Extensión del concepto a un plano

Si se extiende este concepto a un plano cartesiano XY el primer problema que encontramos es el manejo de los dos valores resultantes de dividir los valores de los ejes por un valor L, una forma de representar el sector es mediante el uso de un KD-Tree, la

búsqueda de los sectores contiguos en esta representación tomará entre $\mathcal{O}(log(n))$ y $\mathcal{O}(n)$, dependiendo de la representación del árbol, y para todas las partículas tomará entre $\mathcal{O}(nlog^2n)$ y $\mathcal{O}(knlogn)$ (32). Considerando que se desea mantener todas las partículas en el mismo espacio de memoria debido a la limitada memoria en las tarjetas gráficas, se debe rebalancear el árbol de búsqueda con cada modificación en la posición de las partículas, sin embargo este rebalanceo es demasiado lento ya que la definición de las partículas que residen en la tarjeta gráfica solo puede ser ser realizado por un solo thread que es mas lento en comparación a si se realizara por la CPU.

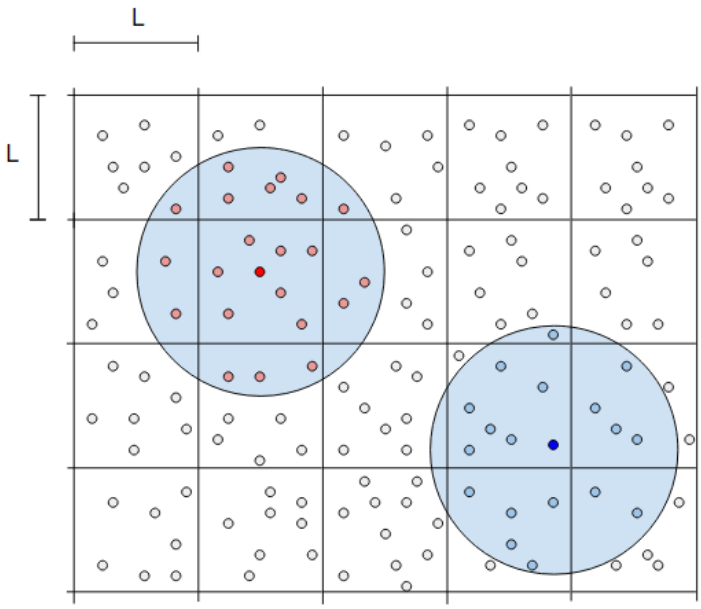

Figure 4.3: El ancho de los sectores tienen el mismo valor que la distancia buscada entre las partículas vecinas, esto implica que las partículas del mismo sector deben ser evaluadas.

Una propuesta para reducir los tiempos de búsqueda es mediante el uso del concepto de la Morton Key, su tarea es encerrar el valor de ambos ejes en un solo valor numérico, con esta llave generada se realiza una búsqueda binaria de tiempo $\mathcal{O}(log(n))$, una vez que las partículas se encuentran ordenadas respecto a esta llave, esto tomará un tiempo promedio de $\mathcal{O}(nlog(n))$ operaciones, al poder implementada en la GPU la ordenación, sin la necesidad de transferencias de memoria a la CPU, así mismo el uso de la Morton Key nos permite utilizar el mismo espacio de memoria donde son almacenas las partículas, sin necesidad de apuntadores de memoria para la construcción de un árbol de búsqueda binario. La explicación de como funciona la Morton Key se detalla mas adelante, esta sección se enfocara en describir como se realizan las búsquedas de las partículas vecinas, sin embargo se debe mencionar que la construcción de la Morton Key se realiza en tiempo constante y que su representación se almacena en una sola

variable numérica.

El segundo problema que surge al extender la idea de una línea a un plano, es que las partículas dentro de un sector de lado L no necesariamente estarán a lo más L de distancia, esto lo podemos comprobar ubicando dos partículas en extremos opuestos del sector, la distancia entre estas será de $\sqrt{2}L$ de distancia, un detalle a considerar para la formulación del ancho del sector. Es en este punto donde la extensión del concepto de una línea a un plano diverge, se decidió no realizar la adecuación del ancho del sector necesaria como se muestra en la figura 4.4, de manera que todas las partículas dentro de un sector no se tengan que revisar, debido a que aumenta los recursos necesarios para ubicar los vecinos.

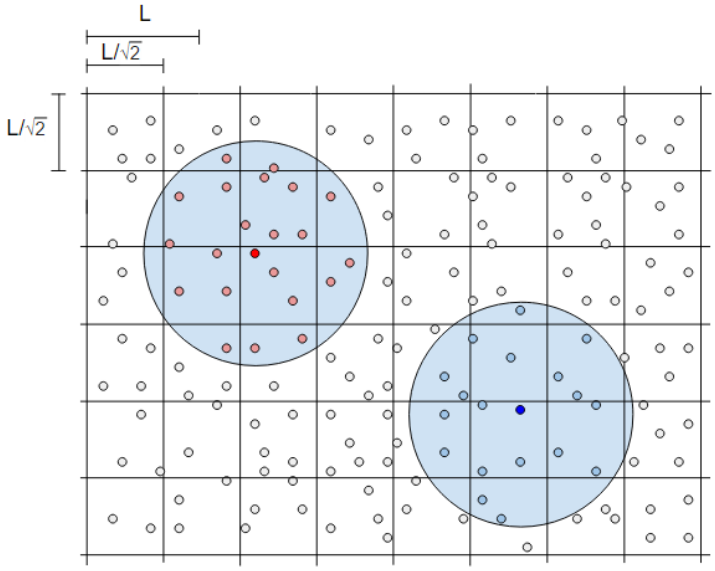

Figure 4.4: El ancho de los sectores es menor al valor de la distancia buscada entre las partículas vecinas, esto implica que las partículas del mismo sector no necesitan ser evaluadas.

El principal problema que surge al definir el ancho de los sector con un valor de $\frac{L}{\sqrt{2}}$ con el objetivo que todas la partículas dentro de este se encuentren a los más L de distancia, da lugar a incrementar la evaluación de sectores de 9 a 20 sectores, y aunque estos son de menor tamaño, implica la evaluación de mas partículas para cada una de las partículas dentro del sector evaluado. Una forma de evaluar el desperdicio de recursos que presenta esta adecuación en el tamaño de los sectores es suponer que cada uno de estos está completamente lleno de partículas, a tal grado que no hay lugar alguno dentro de estos que no señalemos en el cual no exista partícula alguna, por lo que podemos ignorar el concepto de partículas y utilizar el concepto de área del sector.

Cuando se evalúan 9 sectores, todas las partículas viables que afectaran a una

partícula arbitraria en el sector evaluado estarán en un área de $L^2(5+\pi)$ como se muestra en el sombreado de color azul de la figura 4.5, mientras que el área donde no es posible que una partícula afecte a alguna partícula dentro del sector evaluado es de $L^2(4-\pi)$, es decir la parte sombreada de color gris.

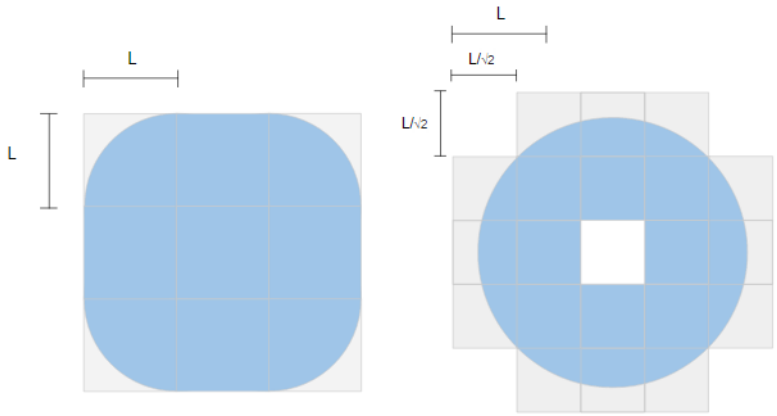

Figure 4.5: Representación de las zonas de búsqueda viables (zona azul) y representación de zonas de búsqueda desperdiciadas (zona gris), donde no es posible que alguna partícula afecte alguna de las partículas del sector central evaluado.

Por otro lado el área efectiva para un sector evaluado cuando se considera la evaluación de 20 sectores, con un ancho de $\frac{L}{\sqrt{2}}$ es de $L^2(9\frac{\pi}{4}-\frac{1}{2})$, considerando que no se realizará la búsqueda en el sector evaluado, lo que da lugar a área de búsqueda desperdiciada de $L^2(\frac{21}{2}-9\frac{\pi}{4})$.

Cuando se procesa una partícula evaluando solo 9 sectores, es decir sectores de L de ancho, solo se desperdicia un área de $L^2(9-\pi)$ de espacio de búsqueda, en comparación con la evaluación de 20 sectores donde se desperdicia un área de $L^2(10-\pi)$, esta comparación en tamaño de sectores se muestra en la figura 4.6.

Esta diferencia puede no significar nada en la elección del tamaño de los sectores e inclusive podemos inclinar la balanza a favor de la elección de sectores mas pequeños, al reducir el área de búsqueda al considerar solo evaluar los 14 sectores que a lo más afectarán una partícula del sector evaluado, reduciendo de esta forma el área de búsqueda a $7L^2$ y en consecuencia el área de desperdicio, que se reduce a $L^2(7-\pi)$, sin embargo esta reducción implica determinar en que cuadrante se localiza la partícula para seleccionar estos 14 sectores afectados, lo que implica un problema en la manera en que se manejan los sectores contiguos en la implementación del código.

No importa que se evalúen 9, 14 ó 20 sectores, la misma búsqueda de sectores contiguos será realizada por varias partículas, es por ello que con el propósito de reducir el tiempo ocupado en el cálculo de los sectores contiguos, estos son precalculados y almacenados antes de realizar las comparaciones de las partículas, lo que significa que

la elección del tamaño de los sectores afectara la cantidad de memoria para reservar de 8 espacios de memoria a 20 espacios de memoria sin importan que solo se utilicen 14 sectores para una partícula determinada, por lo que además de la determinación del cuadrante y la determinación de los 14 sectores, se requerirá mas espacio de memoria para almacenar los sectores contiguos, es por esta razón es que se elige la búsqueda de partículas en sectores de L de ancho, sin mencionar que la implementación de la paralelización en la elección de los 14 sectores causa mas cuellos de botella.

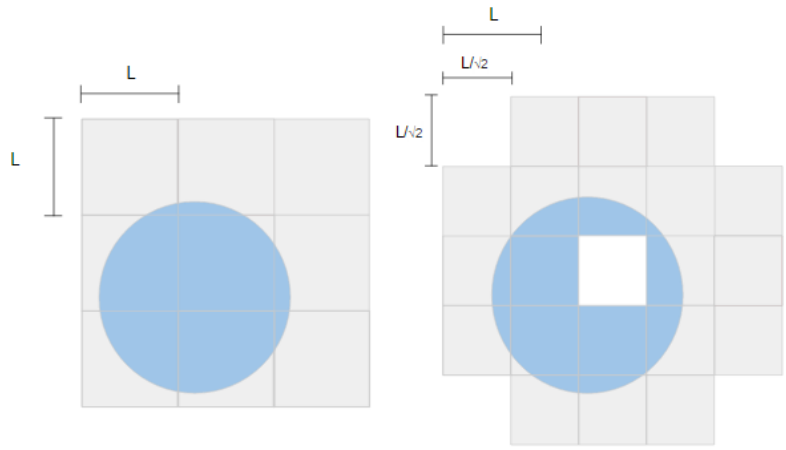

Figure 4.6: La zona azul representa la zona de búsqueda para una partícula del sector central evaluado, la zona gris muestra el área de búsqueda desperdiciada.

4.1.3 Extensión del concepto en 3D

Al extender la representación de un plano cartesiano XY al espacio cartesiano XYZ estos presentas problemas similares a la extensión de una linea a un plano, por ejemplo la representación de los 3 valores obtenidos por las posiciones en el espacio cartesiano sigue siendo comprimida en un solo valor numérico con ayuda del uso de la Morton Key, y de igual forma que con la representación del plano no se considera que todas las partículas deben estar completamente dentro de un sector a al menos L de distancia entre si, debido a las problemáticas que implica su implementación, que aumenta la evaluación de sectores contiguos de 26 a 62, sea que se determine o no el mínimo de sectores afectados por una partícula.

4.2 Morton Key

La idea original detrás de la Morton Key fue desarrollada por G.M. Morton en 1966 (33), una propuesta computacional cuya principal característica es asociar valores numéricos enteros en un solo valor numérico entero, junto con una forma particular de recorrer y encontrar sectores específicos con el uso de máscaras de bits aplicadas a las Morton Key de cada elemento, el uso de operaciones simples como es el recorrimiento de bits y operaciones binarias permite crear y comparar llaves en tiempo constante.

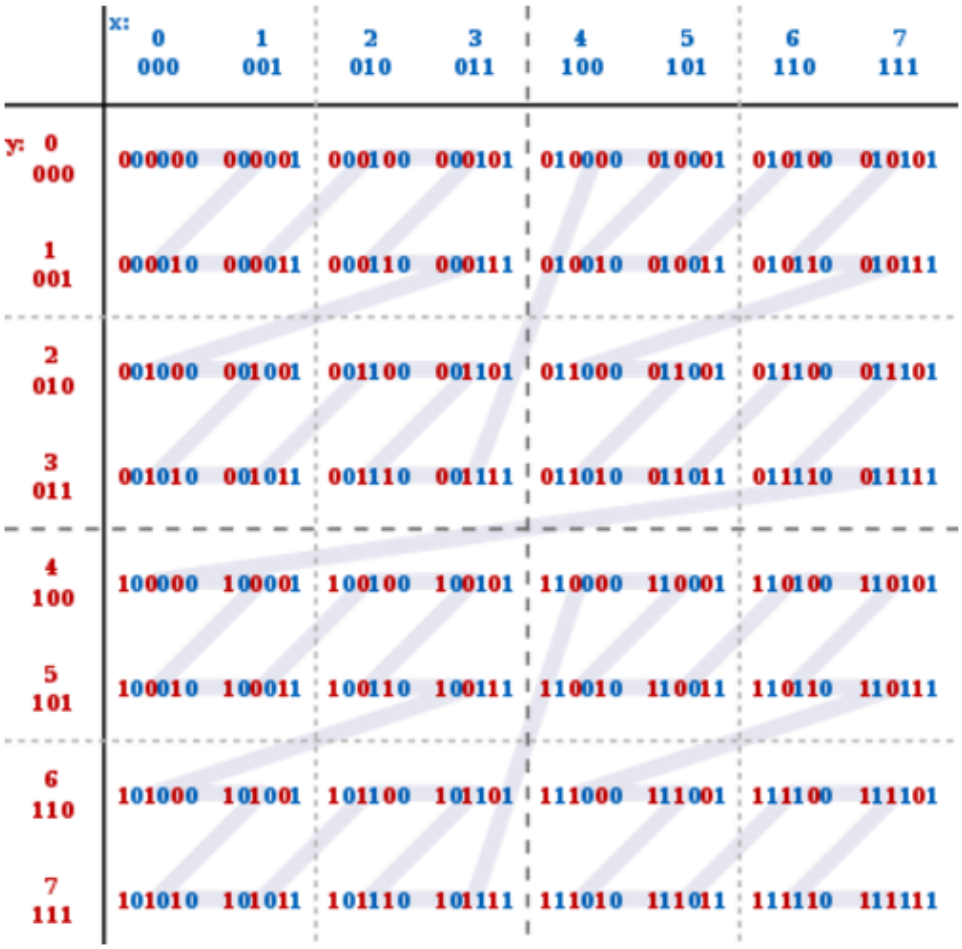

Figure 4.7: Representación de Morton Keys para dos valores, y su seguimiento de manera ordenada considerando el valor numérico obtenido

Una vez obtenidas las llaves para los valores dados es posible organizar y recorrer los datos mediante el uso de máscaras de bits para buscar un sector en particular sin

necesidad de ordenar los valores, esto se logra al aplicar la máscara adecuada a cada uno de los datos, o bien si se ordenan los datos respecto al valor numérico de la Morton Key se pueden recorrer los datos en forma de Z como se muestra en la figura 4.7.

Una de las principales problemáticas que tiene el uso de la Morton Key es que solo es aplicable para valores numéricos enteros, y dependiendo del número de elementos que se quieran agrupar dependerá el rango de elementos posibles que pueden ser utilizados para la Morton Key, al menos en la forma original que fue planteada por G.M. Morton, sin embargo si se considera la forma en que se almacena los números con punto decimal en la computadora (34), podemos hacer uso de la idea de la Morton Key para realizar una composición de los valores para crear una Morton Key para valores flotantes, que será limitada en los posibles valores que puede agrupar, una idea que se descarto debido al limitado rango de valores que se podrían agrupar y su poca viabilidad.

Esta limitante por parte de la Morton Key no es un problema en la presente implementación debido al uso de sectores, que determina para cada partícula en el espacio real ($\mathbb{R}$) la obtención de 3 valores numéricos enteros ($\mathbb{Z}$), los cuales son utilizados para la determinación de la Morton Key, que determina un sector único para cada partícula, sin embargo no todos los valores obtenidos pueden ser utilizados para la creación de la Morton Key, solo los valores mayores o iguales a cero son considerados, esta limitación es necesaria debido a la forma en que se representan los valores negativos en las computadoras, las cuales utilizan complemento a dos en la mayoría de las arquitecturas comerciales (35), por lo que la composición de los valores para formar la llave solo considera un menor rango de números ($\mathbb{N}^+$), sin embargo esto no implica que se debe modificar el valor de la posición de la partícula, solo el valor de la Morton Key que considera un offset global, de esta forma se evita que dos valores numéricos diferentes representen el mismo valor en uno o varios ejes, que podría llevar a la evaluación de partículas que jamas estarán a L de distancia, un problema que puede ocurrir debido a la forma en que es almacenada la Morton Key en una variable entera de 32 bits, ya que para cada eje solo se toman los primeros 10 bits menos significativos. Esto lo podemos ver mas claramente con un ejemplo, imagine dos partículas en el espacio cartesiano XYZ, una ubicada en el punto (11.78, 12.8, 15.2) y otra en el punto (-10230.5, -10235.1, -10238.4), si se considera un ancho de sector $L = 10$ los primeros tres valores obtenidos para la primera partícula sera (1, 1, 1), mientras que para la segunda partícula serán (-1023, -1023, -1023), aunque nosotros sabemos que son dos valores diferentes si se trata de almacenar la Morton Key en un variable entera de 32 bits, y destinamos 10 bits para cada eje, como es el caso de esta implementación desarrollada, el truncamiento de los valores obtenidos resultará en la generación de la misma Morton Key para ambas partículas, ya que la representación binaria de -1023 en un entero de 32 bits que utiliza complemento a dos es 11111111111111111111110000000001 que al tomar los últimos 10 bits representara el mismo valor binario que la primera partícula.

Con el valor obtenido de la creación de la Morton Key para una partícula dada utilizando su posición en el espacio, y la facilidad de aplicar operaciones binarias para determinar sectores contiguos, no se debe caer en la tentación de aplicar a todas las partículas una máscara determinada para encontrar todas las partículas para un sector en específico, ya que se caeríamos en una búsqueda exhaustiva de uno contra todos, sin importar que esta búsqueda exhaustiva se realice en la GPU por varios threads a la vez, así mismo no se hace uso de las Morton Key para la construcción de un árbol de búsqueda, debido a la limitada memoria que poseen las tarjetas gráficas actualmente (Apéndice A), y los tiempos requeridos para la modificación del árbol de búsqueda en cada iteración de la simulación, principalmente debido a los tiempos de accesos para leer y seguir las trayectorias del árbol que causaría cuellos de botella, y una respuesta lenta por parte de la simulación, como lo demostró pruebas iniciales.

La solución que se propone es utilizar el valor numérico obtenido de la Morton Key con el propósito de ordenar las partículas respecto a este valor, de esta forma la búsqueda de los sectores se realizara sobre los índices de la lista de las partículas, de esta forma todas las partículas del mismo sector serán contiguas, esto permitirá la paralelización del código evitando condiciones de carrera, al permitir que cada thread conozca la partícula que debe evaluar casi de manera inmediata, sin necesidad que un thread compita con otro thread por los mismo recursos.

4.2.1 Utilización de la Morton Key para la Búsqueda de Vecinos

Antes de realizar la búsqueda de los sectores vecinos se reduce el espacio de búsqueda creando dos arreglos auxiliares, en el primero se almacenan todos los Valores únicos de las Morton Keys calculas para cada partícula, para el paso actual de la simulación, mientras que en el segundo arreglo se almacenan todas las posiciones iniciales donde las partículas con la misma Morton Key comienzan, el tamaño de ambos arreglos es el mismo y el tamaño de estos dependerá de la simulación, al depender del número de sectores existentes en cada iteración, que puede variar de 1 a N, donde todas las partículas se encontraran en el mismo sector ó cada partícula se encuentra en un sector diferente, este último escenario es poco probable que ocurra sin embargo es necesario considerarlo en la declaración de memoria a utilizar de lo contrario pueden ocurrir posibles errores de acceso de memoria.

La creación de estas lista es de orden $\mathcal{O}(n)$, debido a que se crean las listas después de ordenar las partículas, sin embargo debido a que se realiza la creación de estas listas en las tarjetas gráficas el tiempo sera mas lento, respecto a si se crean en la CPU ya que la creación de estas la realiza un solo thread y no hay forma de distribuir la creación de estos arreglos auxiliares, este pequeño problema no es posible resolverlo de momento y continuara mientras los tiempos de procesamiento para la CPU sean mayores respecto a la GPU haciendo uso de un solo thread.

Una vez creadas las listas auxiliares, estas nos permitirán realizarán la búsqueda de los vecinos mediante una búsqueda binaria $\mathcal{O}(log(M))$, donde M esta entre 1 y

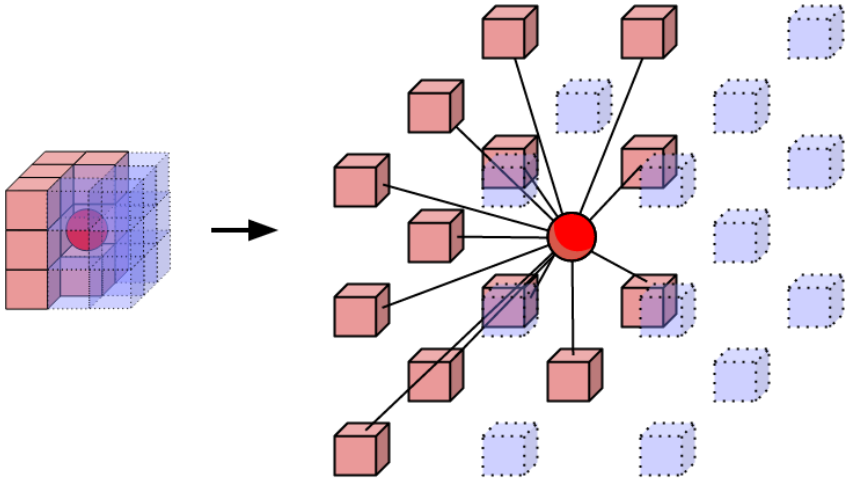

Figure 4.8: Visualización de los sectores contiguos (rojos) respecto al sector evaluado (esfera roja), los sectores restantes (azules) procesarán al sector evaluado cuando sea su turno de ser procesados.

N, y M define el número de sectores para el instante de simulación actual. Para una simulación realista se espera que existan varias partículas vecinas, para poder estimar correctamente los campos de valores buscados, por lo que se espera que varias partículas realicen la misma búsqueda varias veces, es por ello que se guarda para cada sector un arreglo auxiliar los 13 sectores que serán evaluados, los valores almacenados representan el inicio de las partículas a ser evaluados, de esta forma los accesos a los sectores contiguos es de tiempo constante $\mathcal{O}(1)$, mientras que el cálculo de las vecindades es de de orden $\mathcal{O}(Mlog(M))$.

La razón por la que se guardan solo 13 sectores contiguos y no 26 sectores como se planteo en la sección anterior, es debido a que el espacio de búsqueda para cada partícula es simétrico, es decir si una partícula esta lo suficientemente cerca para afectar a la partícula evaluada significa que esta misma partícula también lo estará para la partícula evaluada, por lo que solo sera necesario determinar la distancia L para la mitad de las partículas y procesar los valores de ambas partículas a la vez, de esta forma los 13 sectores contiguos no evaluados sera procesado por algún sector a la "derecha", de existir este sector, de lo contrario significa que no existe partícula alguna que afecte al sector no evaluado (figura 4.8).

Este mismo criterio de reducción de evaluaciones es aplicado también cuando son evaluadas las partículas del mismo sector, como se muestra en la figura 4.9. Cada partícula ocupa un espacio en el arreglo ordenado respecto a la Morton Key, cuando

una partícula evalúa su propio sector para determinar la distancia de las partículas del mismo sector, solo se evaluará las partículas cuyo índice en el arreglo sea mayor, de esta forma si existe otra partícula que afecte a esta partícula, esta será procesada por otra partícula con un índice menor, esta optimización solo es valida siempre y cuando la afectación de las partículas vecinas sea de manera simétrica, de otra forma el planteamiento de los sectores contiguos deberá ser modificado.

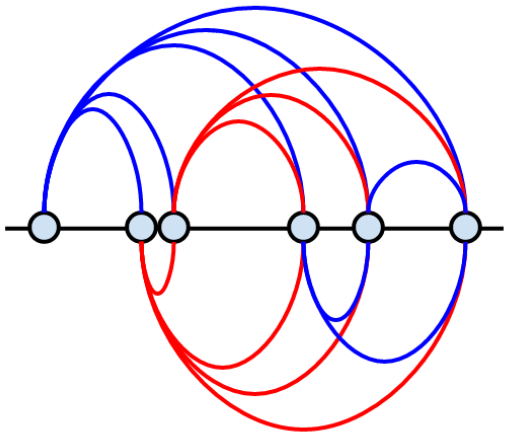

Figure 4.9: Visualización de la evaluación de las partículas dentro del mismo sector para el caso de una recta

4.2.2 Ventajas y Limitantes

La asociación de los valores obtenidos para cada eje están limitados por la unidad de almacenaje utilizada, para el caso tridimensional desarrollado, los tres valores se guardan en una variable de tipo entero INT, representada en la máquina por 32 bits, al repartir equitativamente para cada eje la capacidad de almacenamiento en 10 bits, se limita el número de Morton Keys únicas a 1024^3 sectores únicos, pero limitado a 1024 sectores en cada dimensión, esta limitación dependerá del problema a resolver, por otro lado si se conoce la naturaleza del problema es posible dedicar mas bits para la representación de mas sectores en el eje que así lo necesite o de ser necesario realizar una composición de varias variables, para representar la Morton Key, esta limitante dependerá del valor L elegido para definir los tamaños de los sectores y del dominio del problema a resolver.

En esta implementación se ignora el bit del signo y el bit contiguo a este, el bit mas significativo es utilizado para descartar las partículas que van a ser enviadas a otros nodos, dentro de la misma máquina, pero en tarjeta gráfica diferente o en otras máquinas conectada al sistema CONCORDIA, esto es posible ya que al ordenar las

partículas respecto a los valores generados por la Morton Key en orden ascendente, el activar este bit causará que el valor de la llave resulte en un valor numérico muy grande, lo que nos permitirá determinar que todas las partículas después de cierto índice ya no serán procesadas, así mismo nos permite saber los espacios de memoria disponibles para almacenar las partículas que llegan de otros nodos conectados, otra ventaja que se tiene es el poder hacer una transferencia directa de memoria entre el CPU a la GPU, a partir de un índice conocido, con ayuda de métodos de la API de CUDA, sin necesidad de hacer una búsqueda, para determinar las partículas que ya no serán procesadas. Por lo que tener un bit reservado para descartar las partículas, permite determinar del número de espacios de memoria disponibles de manera inmediata, mientras que el bit de signo nos permitirá ubicar partículas que no queremos que sean evaluadas, esto es de utilidad en el uso de las partículas fantasmas, que requerimos que interactúen con el fluido sanguíneo, pero no modifiquen su posición.

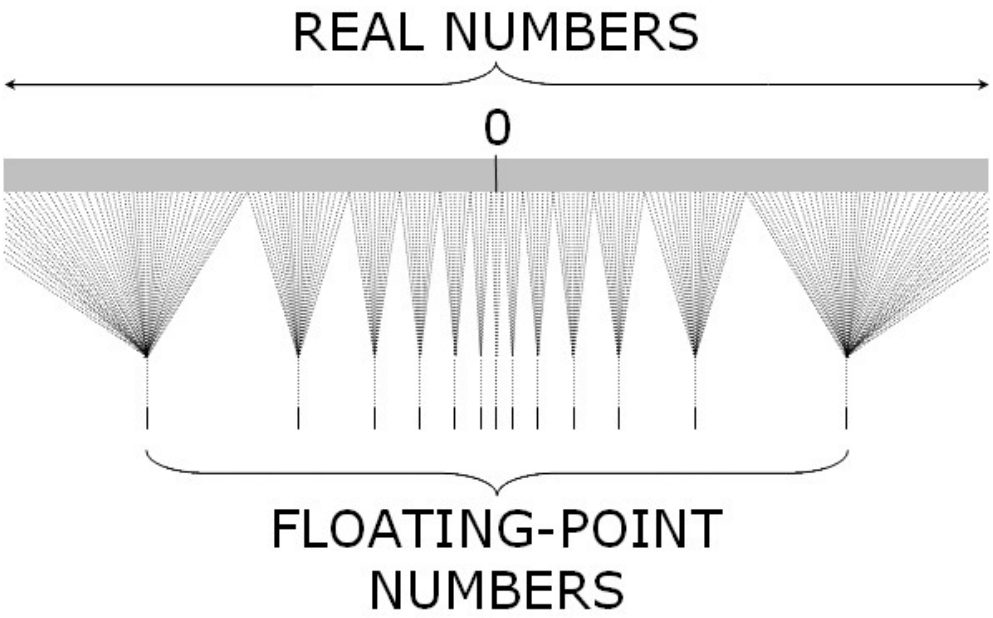

Figure 4.10: Visualización del problema de resolución para el uso de variables tipo *FLOAT*

Una ventaja que tiene en el uso de la Morton Key es la facilidad de poder mover la posición de las partículas, al cambiar directamente el valor de la Morton Key sin alterar los valores de posiciones de las partículas, de esta forma es posible manejar todas las partículas con valores de posiciones relativamente bajos, de esta forma se evita que la representación computacional de los valores altere los resultados, debido a como los valores numéricos pierden precisión debido a la forma en que se representan en la computadora (34), lo que significa que sectores en la frontera con otros nodos que

no necesariamente tienen el mismo desplazamiento requerirán de una variable para la representación de la posición mas grande, a pesar de este problema en la representación del desplazamiento de las partículas, esta propuesta permite el ahorro de espacio de memoria al reducir el número de partículas con variables de alta precisión para almacenar su posición, esta última consideración es necesaria debido al sistema distribuido de máquina utilizado para la distribución de las partículas.

Para la siguiente sección se espera que el lector tenga una noción básica del cómputo en paralelo, para poder ver las ventajas y desventajas que se tienen en la implementación desarrollada, una descripción básica de esto puede ser encontrada en el Apéndice B.

4.3 Procesamiento de las partículas vecinas

La forma en que se propone procesar cada partícula es mediante el lanzamiento de todos los posibles bloques, igual al número de partículas que se pretenda procesar, con el máximo número de threads posibles, que se tienen disponibles para cada tarjeta gráfica, de esta forma cada bloque evalúa una partícula mientras que los threads cooperan entre si para evaluar la misma partícula, al estimar la distancia y ponderando cuanto afectan a la partícula evaluada las partículas ubicadas en los 14 sectores vecinos, para calcular la propiedad deseada. De estos sectores, 13 son debidos a los sectores contiguos y uno es donde se encuentra la propia partícula evaluada (figura 4.8).

La manera en que se determina para cada thread cuales partículas se deben procesar para cada sector contiguo es determinado de manera casi inmediata sin que ocurran condiciones de carrera, esto es logrado utilizando el identificador que cada thread posee, y aprovechando que las partículas ya se encuentran ordenadas respecto a su Morton Key, cada thread avanza sobre los índices del arreglo en lugar de seguir apuntadores de memoria, esto lo podemos ver mas claramente con un ejemplo: En una simulación con 2050 partículas la primer partícula a ser procesada, determina el sector al que pertenece y sus sectores vecinos, para este ejemplo el primer sector contiguo a procesar inicia en la posición 0 del arreglo donde se almacenan las partículas, este sector contiguo contiene 200 partículas, por lo que el thread con el identificador 250 determina de manera segura que la partícula que le corresponde evaluar no se encuentra en el primer sector contiguo, por lo que procede a evaluar el segundo sector en el que probablemente se encontrará la partícula 250 a ser evaluada, de esta forma para un bloque que cuenta con 1024 threads, el thread 0 evaluara la partícula 0, 1024 y 2048, de esta forma los threads no requieren esperar a ningún thread para continuar su procesamiento, por lo que procede a la siguiente partícula sin necesidad de esperar o informar a nadie.

De esta forma si se lanzan 500 bloques, el primer bloque con el identificador 0 evaluará la partícula 0, 500, 1000, 1500 y 2000 y el thread 0 del bloque 0, evaluará las partículas vecinas 0, 1024 y 2048 del las partículas 0, 500, 1000, 1500 y 2000 respectivamente, las posiciones de las partículas vecinas a ser evaluadas no necesariamente serán las mismas partículas, estas partículas vecinas estarán determinadas por los sectores contiguos para cada sector evaluado.

Suponga un ejemplo mas simple donde la Morton Key se compone de 6 bits, y el valor resultante de cada dimensión es almacenado en 2 bits de esta llave, este ejemplo se tiene 21 sectores con 10 partículas dentro de cada uno de estos, la representación de estos 21 sectores se presenta en la siguiente tabla, donde la primera fila representara el valor numérico de la Morton Key, la segunda fila representa su forma binaria, y la tercer fila el inicio donde las partículas con la misma Morton Key comienzan.

0	4	12	16	20	28	48	52	60	1	5	13	17	21
000000	000100	001100	010000	010100	011100	110000	110100	111100	000001	000101	001101	010001	010101
0	40	120	160	200	280	480	520	600	10	50	130	170	210

En este ejemplo en particular se lanza un kernel con 100 grids de 20 threads cada uno, de esta forma, el thread 0 del bloque 21, evaluará la partícula que esta en la posición 0, al terminar de procesarla evaluará la partícula 120, esto es debido a que los 20 threads lanzados en su primera iteración se encargaran de procesar las partículas 0 a 9 y 40 a 49, en su segunda iteración procesaran las partículas 120 a 129 y 160 a 169, de esta manera cada thread sabe inmediatamente que partícula debe procesar, mientras que el los sectores que evaluaran los bloques 0 a 9 , serán los siguientes:

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0
													000000
													0

Para el Grid 0, solo los primeros 10 threads tendrán un carga de trabajo y evaluarán solo las partículas que están en su mismo sector, en particular para la implementación desarrollada la forma de determinar si un sector en valido o no, es mediante la Morton Key que debe tener un valor numérico positivo.

La forma en que se evito la sincronización explicita para los threads, es mediante el uso de memoria auxiliar que permite que cada thread guarde el valor calculado en un espacio de memoria, y al terminar de procesar todas las partículas, los valores calculados son sumados, y almacenados en su localidad de memoria, esto sin embargo implica que para cada partícula se requieren tantos espacios de memoria como threads sean lanzados, esta propuesta presenta los mejores tiempos de respuesta debido a que no requiere la implementación de candados virtuales, ya que no se asegura que no existan condiciones de carrera al momento de guardar los valores calculados, debido a que los threads cooperan para procesar una partícula a la vez, y no pueden avanzar hasta terminar de procesar esta partícula, lo que trae con sigo un cuello de botella importante en los cálculos.

Una propuesta que se descarto fue el uso de la memoria compartida por los bloques, ya que no permite implementar la optimización de solo evaluar la mitad de las partículas, ya que el uso de esta obliga a cada partícula evaluar cada una de las partículas en los 27 sectores que afectan a la partícula evaluada, ya que no es posible almacenar el cálculo de la partícula encontrada, además de esta doble búsqueda de partículas que se encuentren a L de distancia es necesario una barrera de sincronización, con el objetivo de asegurar que cada thread agregue el valor calculado en la memoria compartida para la partícula evaluada, este escenario lo podemos ver reutilizando el ejemplo anterior, en la evaluación del bloque 0, solo 10 threads tendrán una carga de trabajo mientras que el resto tendrá que esperar a que estas 10 partículas sean procesadas, para finalmente guardar el valor de la propiedad calculada en la partícula procesada, a diferencia de la propuesta implementada donde el uso de la memoria auxiliar permite que el resto de los threads continúen efectuando los cálculos de las partículas correspondientes sin necesidad de esperar o informar que ya termino su carga de trabajo.

4.4 Manejo de las Fronteras

El manejo de las fronteras y en especifico para el caso de estudio propuesto, se simulan mediante el uso de método híbrido donde se definen una serie de capas de partículas fantasma que contiene el flujo de la simulación, estas capa de partículas fantasma están atadas entre si y se mueven en conjunto para simular la contracción y dilatación de las arterias, así mismo estas partículas actúan como sensores para determinar la fuerza con que el fluido empuja las paredes arteriales, la primer capa de partículas interactúa con el flujo evitando que estas salgan de las fronteras definidas y como sensores de presión, mientras que la segunda capa es usada como respaldo por si alguna partícula escapa de la primera barrera de contención, esta segunda capa no interactúa con el flujo de ninguna manera y solo ayuda a mantener las partículas dentro de las paredes arteriales.

El manejo de las fronteras como partículas, son tratadas de manera similar que las partículas del flujo, solo que estas se encuentran fijas y se mueven de manera arbitraria para formar las paredes arteriales, además del uso de partículas fantasma se utilizan geometrías básicas definidas por el usuario (cilindro, paralelepípedo y esfera) que permiten limitar el flujo de mejor manera.

La primer propuesta requiere determinar la ubicación de las partículas fantasma en la geometría definida por el usuario, y para este caso en particular se debe considerar la dilatación máxima y la contracción para evitar que las partículas del fluido puedan escapar, la segunda propuesta de solución requiere especificar los parámetros dependiendo de la forma geométrica elegida, en la presente implementación se utilizan ambas propuestas para definir las paredes de los vasos sanguíneos.

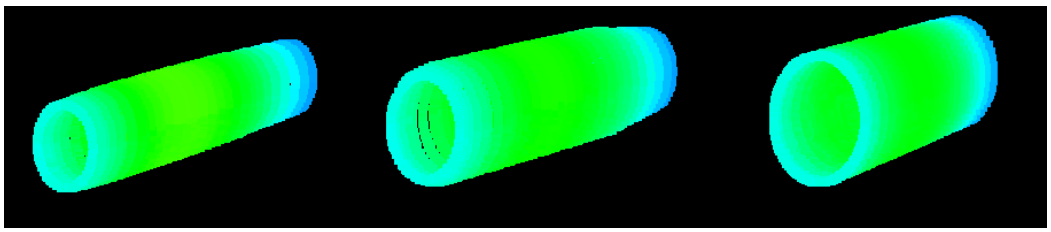

Figure 4.11: Segmento de Arteria representado por la primer capa de partículas fantasma, a diferentes paso de la contraccion y dilatación

La solución para resolver el problema de las fronteras involucra las utilización de ambas propuestas, el uso de las partículas fantasma permite definir las paredes arteriales y simular la vasoconstricción y vasodilatación, mientras que el uso de primitivas geométricas permite representar la cánula que sera utilizada para colocar el stent en su lugar, debido a que su forma simple permite representar la cánula como un cilindro.

4.5 Ordenación de las Partículas

Al comienzo del desarrollo de la simulación se contemplo la utilización de la API de Thrust, con el propósito de utilizar los métodos de ordenación implementados en esta API, y poder ordenar las partículas respecto al valor obtenido por la Morton Key, sin embargo después de unas pruebas, el método encargado de ordenar las partículas mostraba un error de acceso de memoria, una falla que era completamente ajena a la implementación de la simulación, por esta razón se desarrollo un método híbrido de ordenación, que manejara los recursos de memoria de mejor manera, el método implementado hace uso del método bitonic sort (36), así como una propuesta para unir dos listas ordenadas en una sola lista ordenada, el número de elementos a ordenar no esta sujeto que sea potencia dos, esto se logra al efectuar dos veces la ordenación de la lista, primero al ordenar una parte de la lista, considerando que el número de elementos a ordenar debe ser la potencia dos mas cercana y menor al número de elementos de la lista,

una segunda ordenación se efectúa sobre los elementos restantes, considerando de igual forma que el número de elementos a ordenar debe ser potencia de dos, considerando que se debe ordenar los elementos no utilizados en la primera ordenación, sin importar que se utilicen nuevamente elementos previamente ya ordenados, esto lo podemos ver con un ejemplo, en un arreglo de 23 elementos, la primera ordenación se realizara con los primeros 16 números de la lista, la siguiente ordenación sera a partir del elemento numero 14, y se ordenaran 8 elementos, 5 elementos no utilizados previamente y 3 elementos que nuevamente se ordenan en la segunda lista, finalmente se hace la mezcla de ambas listas para obtener una sola lista ordenada, es costo de hacer es de orden:

$$\mathcal{O}(log^2 Q) + \mathcal{O}(log^2(R)) + \mathcal{O}(log(Q)) + \mathcal{O}(log(R))$$

Donde $Q = \lceil log_2(n) \rceil$, y R = $\lceil log_2(n - m) \rceil$, los primeros dos términos definen los tiempos que toma realizar las ordenaciones dentro de la GPU, los siguientes dos términos definen los tiempos de intercambio que toma juntar ambas sublistas en una sola lista ordenada, este intercambio de lugares se realiza mediante una búsqueda binaria para cada uno de los elementos de la lista contraria, de tal forma que cada elemento busca su posición a ocupar en la lista contraria.

Esta propuesta no esta exenta de necesitar un espacio de memoria para realizar los intercambios, sin embargo la propuesta presentada no requiere de un espacio de memoria igual al número de elementos de ambas lista, solo de un espacio de memoria para realizar el intercambio, así mismo se necesita de un arreglo igual al número de elementos, en este se almacenara el número de elementos que se debe mover un elemento a la izquierda o a la derecha para ocupar su lugar final, esta lista ocupara un menor espacio de memoria ya que solo especifica un valor numérico, y no la definición completa de las partículas (posición, masa, densidad, velocidad, etc), esta propuesta además de permitirnos ahorrar espacio de memoria para almacenar los intercambios muestra ser un método estable y confiable.

El método propuesto puede ser visto mas claramente con un ejemplo, suponga una lista de 14 elementos:

3	6	2	5	56	436	2	57	2	2	65	1	35	7

La primera ordenación se realizara sobre los primeros 8 elementos, resultando de la siguiente manera:

2	2	3	5	6	56	57	436	2	2	65	1	35	7

Mientras que la segunda ordenación utilizara los últimos 8 elementos de la lista, ordenando los 6 elementos que no se tomaron en la primera ordenación y volviendo a utilizar los últimos 2 elementos de la primera ordenación, la lista se vera de la siguiente manera:

2	2	3	5	6	56	1	2	2	7	35	57	65	436

Después de realizar ambas ordenaciones, se busca para cada elemento de ambas sublistas cuantos elementos deben desplazarse, a la izquierda o a la derecha, para que el elemento se ubique en su posición final, para el primer elemento de la sublista de la izquierda, este se debe moverse 3 espacios a la derecha, ya que se van a colocar los primeros tres elementos de la sublista de la derecha antes de este, la determinación de cuantos elementos se debe mover cada elemento es posible realizarse recorriendo de principio a fin la sublista contraria, o de manera mas óptima descartando mitades de la lista, como si se tratara de una búsqueda binaria, aprovechando que los elementos de ambas sublistas ya se encuentran ordenadas, la sublista de la izquierda debe encontrar el número de elementos que son menores o iguales a este elemento evaluado, de manera similar cada uno de los elementos de la sublista de la derecha realiza la misma búsqueda considerando solo los elementos menores no iguales, utilizando este principio los tres primeros elementos de la sublista de la derecha debe moverse 6 elementos a la izquierda, por lo que para el ejemplo anterior el desplazamiento de cada elemento queda de la siguiente forma:

2	2	3	5	6	56	1	2	2	7	35	57	65	436
+3	+3	+3	+3	+3	+ 5	-6	-6	-6	-1	-1	0	0	0

Los intercambios de los elementos se pueden realizar inmediatamente si se tiene una lista auxiliar del mismo tamaño que definen los elementos, lo cual no es recomendable ya que ocupa el mismo espacio de memoria que la lista que se trata de ordenar, lo mas recomendable es efectuar un intercambio a la vez siguiendo las trayectorias de los desplazamientos, ya que ningún elemento de intercambio choca con otro elemento de la propia sublista o de la sublista contraria, tal como se muestra en la siguiente tabla. Para poder visualizar mejor los elementos de intercambio, estos son mostrados en filas diferentes en sus ubicaciones finales.

2	2	3	5	6	56	1	2	2	7	35	57	65	436
+3	+3	+3	+3	+3	+ 5	-6	-6	-6	-1	-1	0	0	0
			2	2	3	5	6			56			
1	2	2						7	35		57	65	436

Un ejemplo para una trayectoria de desplazamiento se muestran en la siguiente tabla con los elementos 2 y 5 de la sublista de la izquierda y el elemento 1 de la sublista de la derecha.

2	2	3	5	6	56	1	2	2	7	35	57	65	436
+3	+3	+3	+3	+3	+ 5	-6	-6	-6	-1	-1	0	0	0

Estos elementos forman un ciclo por lo que podemos aprovechar sus espacios de memoria para realizar los intercambiados, siguiendo esta trayectoria el elemento 2 debe ocupar la posición del elemento 5, el elemento 5 debe ocupar la posición del elemento 1 y finalmente el elemento 1 debe ocupar la posición del elemento 2. El número de elementos para estas trayectorias de intercambio, así como el número de elementos en estas dependerá de los elementos que contengan las sublistas, donde el mejor de los casos es cuando no se tiene que realizar intercambio alguno, y el peor de los casos, es cuando se tiene que realizar un intercambio completo de la lista de menor tamaño. Esta solución propuesta, para la ordenación de los elementos fue implementada en CUDA y se optó por seguir mas de una trayectoria a la vez, dejando que cada thread siga una trayectoria de intercambio, de esta forma es posible terminar de unir ambas listas en un menor tiempo, respecto a si solo lo realizara un solo thread.

La forma en que se determina las trayectorias únicas de intercambio, es calculando para cada elemento el índice del elemento con que se pretende intercambiar, colocando el menor índice de estos dos elementos, de esta manera se determinaran todas las trayectorias y se determinara donde comienzan cada una de estas trayectorias, al saber que la partícula que tiene su mismo índice es el inicio de los ciclos de intercambio, utilizando el ejemplo anterior la primera fila indican los índices en el arreglo que se agregan para visualizar los intercambios de índice y la última fila el arreglo auxiliar después de procesar cada elemento.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
2	2	3	5	6	56	1	2	2	7	35	57	65	436
+3	+3	+3	+3	+3	+ 5	-6	-6	-6	-1	-1	0	0	0
0	1	2	0	1	2	0	1	2	8	5	11	12	13

De esta forma determinamos que solo los elementos cuyo valor auxiliar sea igual al índice del elemento serán los inicios de las trayectorias de transferencias, que para este ejemplo son los primeros tres elementos de la sublista de la izquierda, el número máximo de intercambios estará en función de $R = (\lceil log_2(n-m) \rceil)$, que es el número de elementos en la sublista, donde el peor escenario posible requiere de $(n - 2^{\lceil sqrt(n) \rceil - 1}$) espacios de memoria, este escenario se presenta cuando todos los elementos deben ser intercambiados y el tamaño de las trayectorias de intercambio es de solo dos elementos.

4.6 CONCORDIA

Antes de iniciar descripción de la implementación del sistema CONCORDIA, es necesario tomarnos un momento para reflexionar y considerar los elementos involucrados en

la creación de este, es por ello que la siguiente subsección es una breve descripción del inicio de la Internet y las tecnologías que hicieron posible la conexión entre diferentes máquinas, para crear el sistema que permite la utilización de recursos compartidos, usados en la implementados de esta tesis.

4.6.1 Antecedentes

En 1957 con el lanzamiento del satelite sputnik, por parte de la Unión Sovietica el presidente de los Estados Unidos Dwight D. Eisenhower, crea en 1958 la Agencia de Proyectos de Investigación Avanzada (ARPA) con el único propósito de avanzar de manera significativa los conocimientos científicos de esta nación, esta nueva agencia tenia como propósito inicial la detección de pruebas nucleares y la posible defensa ante ataques de misiles balísticos intercontinentales por parte de la unión Soviética, y fue en esta agencia, bajo la dirección de J.C.R. Licklider que en 1962, la idea de la interconexión de máquinas para la compartición de recursos que dio inicio.

A principios de 1960 Licklider, Leonard Kleinrock, Paul Baran, Lawrence Roberts, junto con varios investigadores empezaron a trabajar con la idea de la creación de una red global donde cualquier computadora pudiera accesar otra desde cualquier parte del mundo, esta idea finalmente se materializo en la creación de la Internet que es usada hoy en día. Pero no fue sino hasta 1970 cuando ARPANET fue establecida, contando en su poder la conexión de 12 máquinas interconectadas a lo largo de todo Estados Unidos, incluyendo las máquinas en el centro de operación de Redes (NCC) en la Corporación de Tecnología Bolt Beranek y Newman en Harvard, y la RAND (Corporation and the Massachusetts Institute of Technology). Esta red contaba con la incorporación de la nueva tecnología que permitía la transferencia de archivos por medio del Protocolo de Transferencia de Archivos (File-Transfer Protocol, FTP) propuesto por la RFC 354 en Julio de 1972 y fue en Octubre de 1972, cuando la idea de interconexión de máquinas para uso militar, dejo de ser uso exclusivo del ejercito de los Estados Unidos, cuando se hace pública esta idea revolucionara en la Conferencia de Comunicación en Computadoras, y fue en esta misma conferencia donde se realiza el primer envío de correo electrónico entre dos computadoras por Ray Tomlinson y uno de sus colegas, no paso mucho tiempo desde esta conferencia, para que esta red se convirtiera en una nueva forma de comunicación, y para 1973 tres cuartas partes del tráfico de la red, fuera en su mayoría correos entre investigadores compartiendo información, para ese entonces los científicos habían estado usando el Protocolo de Control de Red (Network Control Protocol, NCP) para transferir entre computadoras información, pero aun estaban sujetos a tener conexiones continuas para que las transferencias se completaran de manera exitosa, lo que limitaba las transferencias y tamaño de archivos que se compartían, fue esta limitante lo que obligo a buscar alternativas para acelerar la transferencia de recursos, y fueron los investigadores Vinton Cerf de la Universidad de California UCLA y la Universidad de Stanford junto con Robert Kahn de ARPA que desarrollaron el protocolo de Transmisión de paquetes usados hoy en día para la transferencia de información a lo largo del mundo mediante el Protocolo TCP / IP,

a finales de los años 1970. Con el uso del protocolo TCP / IP la interconexión de computadoras se incremento a lo largo del mundo, sin importar la diferencias de red en la que se encontrará, una de las principales limitantes en aquel entonces, debido a las diferencias por parte de los fabricantes que desarrollaban conexiones de red distintas. Así mismo la implementación del protocolo TCP / IP hizo el Internet mas rápido y eficiente al dividir en paquetes la información, lo que permito el crecimiento de la Internet, casi de manera inmediata, con ayuda de otros avances realizados como, el concepto de Ethernet, desarrollado en 1976, por Robert Metcalfe, lo que permitió la transferencia de mayor información a través del uso de cable coaxial, lo que dio lugar al desarrollo del proyecto de empaquetado satelital (SATNET), lo que dio lugar a la conexión entre los Estados Unidos con Europa, concretando la idea de conexión entre máquinas sin importar su ubicación.

Y fue en 1990 cuando Berners-Lee trajo al mundo el concepto "World Wide Web", al escribir la primera pagina html, introduciendo este nuevo concepto en la conferencia Internacional de Diciembre de ese mismo año, utilizado esta nueva forma de presentar los datos un año después por el público en general. Fue en 1990 que ARPANET fue decomisionada, después de 20 años de servicio, la red NSFNET backbone cuya velocidad era 25 veces mas rápida que ARPANET, tomo la democratización de la red mas rápido de lo esperado, y un año después, gracias a la facilidad de uso de la idea desarrollada por la red de Berners-Lee, y los proveedores de servicio (ISPs), se comenzó a comercializar el acceso a Internet y para 1995, la Internet tenia un estimado de casi 16 millones de usuarios y negocios, que sigue creciendo hoy en día.

Desde 1991, con la facilidad de mostrar información y de conectar personas y negocios a lo largo del mundo la Internet creció de manera exponencial, y para 1996 aproximadamente 45 millones de personas hacían uso de la Internet, y para 1999 el número de usuarios se acercaba a 150 millones para el año 2000, el número creció mas del doble en un solo año alcanzando casi 407 millones de usuarios en 218 de las 246 naciones en el mundo.

Gracias a la colaboración de varios investigadores en las últimas cuatro décadas el crecimiento de la Internet ha permitido el envío de una variedad de datos, ya no solo se envían correos entre los investigadores, sino también diversa información entre computadoras, materializándose el sueño que inicio en 1962 con J.C.R. Licklider y sus colegas, pero en especial hay que dar un especial agradecimiento a aquellas personas que hicieron posible la conexión entre máquinas que hoy en día usamos, gracias Ivan Sutherland, Robert Taylor, Alex McKenzie, Frank Heart, Jon Postel, Eric Bina, Robert Cailliau, Tom Jennings, Mark Horton, Bill Joy, Douglas Engelbart, Bill Atkinson, Ted Nelson, Linus Torvalds, Richard Stallman, Dave Clark y muchos otros que es casi imposible de mencionar, que sin su ayuda seria posible la utilización del sistema desarrollado en esta tesis.

4.6.2 Descripción de Conexión

Han pasado mas de cuatro décadas para llegar a tener la infraestructura que tenemos hoy en día, con la posibilidad de conectar casi de manera inmediata cualquier dispositivo de electrónico con otro dispositivo en cualquier parte del mundo, lo que se propone en el presente trabajo es la utilización de esta infraestructura que se encuentra a nuestro disposición, para permitir aumentar el poder computacional que se tiene al conectar varias máquinas que no necesariamente se encuentren en la misma red local, permitiendo de esta forma aumentar el número de partículas que es posible procesar a la vez, sin necesidad de realizar una inversión adicional en la compra de nuevas tarjeta gráficas. Actualmente existen una gran cantidad de tarjetas gráfica y cada año son lanzadas al mercado nuevas tarjetas, marginando los recursos que ya se tienen, lo que se propone es utilizar estos recursos que muchas veces no difieren de los nuevos productos lanzados, para poder incrementar la capacidad de cómputo, esto no significa que se deje de invertir recursos en nuevas tarjetas gráficas, sino todo lo contrario, las nuevas inversiones podrá ser añadida a los recursos que ya se tienen para incrementar el poder de cómputo.

La intención es utilizar las tecnologías desarrolladas para los navegadores, para de esta manera transmitir la información que será utilizada por otras computadoras permitiendo cooperar de manera en particular la simulación del fluido biológico de la sangre.

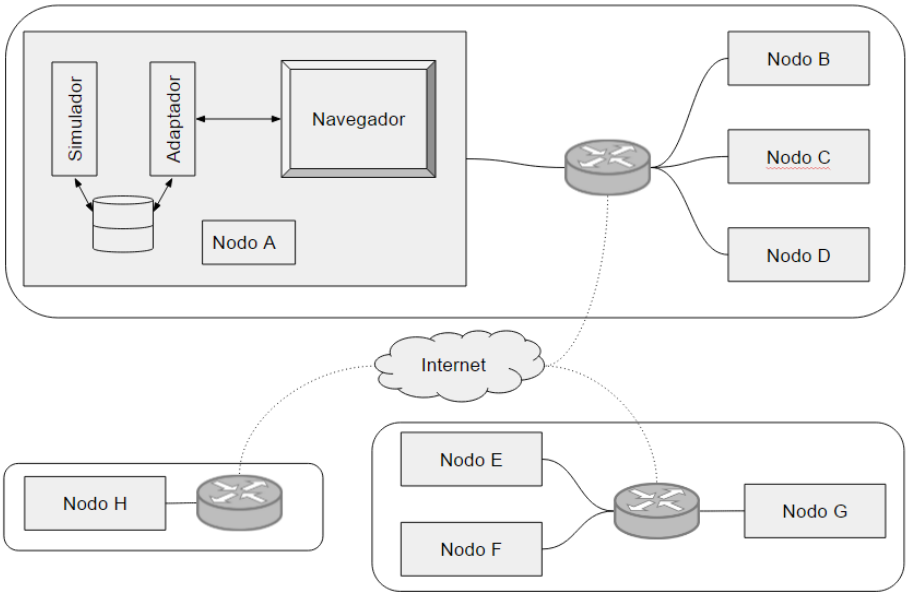

Figure 4.12: Ejemplo de conexión usando CONCORDIA

La forma en que el sistema maneja el envío de los cálculos de las partículas de una máquina a otra es manejado por un programa de ejecución independiente, a la

simulación desarrollada, este programa tiene la tarea de enviar y recibir información mediante un Websocket conectado a un navegador, es necesario que la ejecución de este programa sea independiente al programa que ejecuta la simulación debido a que este debe mantener activa la conexión con el navegador al mandar respuestas a intervalos regulares, de no ser independiente este programa a la simulación realizada, las solicitudes enviadas por el navegador podrían ser respondidas tarde y el navegador cerraría la conexión, un problema que puede ocurrir debido a que no se pueden estimar con certeza los tiempos que tomará realizar los cálculos, que pueden variar dependiendo del estado de la simulación, y de la problemática a resolver.

Una ventaja que tiene la utilización de un programa independiente para enviar y recibir información, es que puede actuar como un adaptador de comunicación entre cualquier aplicación de escritorio con el navegador, esto es posible al crear un espacio de memoria, donde es posible la escritura y lectura por parte del programa que desea conectarse con el navegador y el adaptador desarrollado, este adaptador permite comunicar con facilidad varios lenguajes de programación entre si, en distintas máquinas a lo largo de la Internet, teniendo en cuenta que todas las máquinas conectadas deben tener arquitecturas similares de almacenamiento de información (littlendian o bigendian), o bien contemplar estas diferencias para que todas las máquinas reciban la misma información y puedan manipularla correctamente.

Una vez que la información se encuentra en el navegador es posible enviar y recibir con cierta facilidad información desde y hacia otros nodos, por medio de los navegadores, utilizando los recursos desarrollados para estos, lo que nos permite presentar resultados de la simulación en cualquier navegador, sin la necesidad de instalar algún programa en particular, un ejemplo de esto es la visualización de las partículas mediante la ayuda de la API de Tree.js, apoyándose de la tecnología de WebGL.

La manera en que el navegador se comunica con otros navegadores de otras computadoras es mediante el uso de la tecnología de WebRTC (37), esta permite un intercambio de información entre navegadores de manera fácil y rápida entre los nodos que pretenden compartir sus recursos, esto se realiza mediante una conexión directa entre las computadoras que compartirán recursos sin la necesidad de pasar por un nodo central, esto es posible una vez que se realiza un proceso de señalización que permite que todos los nodos compartan información.

Esta carrera de relevos, por así decirlo, de envío de información entre distintas partes, del simulador al adaptador, del adaptador al navegador y de este a los demás nodos, y nuevamente de regreso, del navegador al adaptador hasta pasar nuevamente de regreso al simulador, es tan rápido como la fase de envío mas lenta, en esta carrera para el envío de la información, la fase mas lenta dependerá de las condiciones de la red en general, que puede ubicar como la conexión entre nodos por medio de WebRTC la fase mas lenta, o bien la comunicación entre el Websocket y el adaptador desarrollado.

4.6.2.1 Primera fase de Conexión

Lo primero que se necesita hacer, es determinar si se necesita enviar los resultados a otra tarjeta gráfica dentro de la misma máquina o bien enviar esta información por la red para que sea procesada por otra máquina. Si no es necesario enviar la información por la red, se procede a utilizar los métodos proporcionados por la API de CUDA, para el envío de la información entre las tarjetas que se ubican en la misma máquina.

El primer eslabón en la cadena para el envío de información es el adaptador que permite enviar y recibir los datos calculados del simulador y mantener la conexión activa con el navegador, esto se logra al comunicar ambos programas mediante un segmento de memoria compartida, utilizando dos buffers circulares, uno de entrada y otro de salida, de esta forma los mensajes son procesados conforme son recibidos y son enviados tal como son procesados, lo que permite que un tiempo de simulación sea procesado mientras otro esta siendo enviado o recibido por la red, de esta forma el procesamiento y envío de la información es ininterrumpido.

Dentro de estos buffers circulares, cada segmento es almacenado con una cabecera que contiene información que sera utilizado por el navegador y por el nodo que recibe estos paquetes de información, esta información incluye el valor de la Morton Key que representa al sector que contiene las partículas a ser enviadas, así como el identificador del nodo al que debe ser enviado. La máquina de procedencia es agregado por el navegador al momento de ser enviado a los nodos indicados, se realiza de esta forma, para que el adaptador no lea ni procesé ningún tipo de información, de esta forma limitamos la función del adaptador al envío y recibimiento de información, lo que permite adecuar cualquier programa para que utilice el adaptador con otros programas que no necesariamente se ubiquen en la misma máquina, así mismo este permite actuar como una puerta de conexión con el navegador, lo que permite establecer mas de una conexión simultanea, no solo con computadoras sino también con dispositivos dedicados, como es el caso de la Arduino y la Raspberry Pi.

La comunicación entre el adaptador y los distintos programas se realiza a través de la escritura y lectura de segmentos compartidos de memoria, el como se lee y escribe en este segmento dependerá de cada lenguaje de programación, en particular para el simulador y el adaptador desarrollados en C++ se hace uso de la API de Boost la librería, *managed_shared_memory* para leer los apuntadores de memoria compartida y de la librería *interprocess_mutex* para controlar la escritura en los sectores de memoria.

4.6.2.2 Segunda parte de Conexión

La segunda parte de conexión desarrollada, es la comunicación entre el adaptador y el navegador, esto es logrado mediante la utilizando la RFC 6455 (38), que define la forma en que cliente y servidor se conectan para crear una canal de comunicación bidireccional, el primer paso que se implemento fue el handshake entre el adaptador y el navegador, un ejemplo de un handshake por parte del cliente, definido por la RFC 6455, se verá de la siguiente forma:

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

Mientras que el handshake por parte del servidor se verá como:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat

Una vez que se inicio la conexión del cliente con el servidor, el cliente debe esperar una respuesta del servidor antes de enviar cualquier información, el cliente debe validar la respuesta del servidor, y una vez validada la respuesta se establece la conexión activa, permitiendo enviar y recibir información inmediatamente.

La misma definición de la RFC 6455 describe la configuración que los paquetes debe tener para definir adecuadamente los envíos de información, lo primero que se debe definir en la cabecera del mensaje es el código de operación (opcode), esta escritura de la cabecera es realizada mediante operaciones básica de recorrimiento de bits, que con ayuda de la API de Boost, permite el envío de estos paquetes a través de la red.

Los posibles valores que puede contener el opcode definido en 4 bits esta definida en la RFC 6455, mostrados a continuación:
%x0 Denota un frame de continuación
%x1 Denota un frame de texto
%x2 Denota un frame binario
%x3-7 Son reservados para futuros frames de control
%x8 Denota un cierre de conexión
%x9 Denota un ping
%xA Denota un pong
%xB-F Son reservados para futuros frames de control

Una vez que el navegador recibe la posición de las partículas así como sus propiedades, el navegador es el encargado de gestionar el envió de cada partícula a las máquina determinadas por la cabecera dentro de los paquetes de información, y si asi se requiere graficar la posición de estas.

La manera en que el nodo maestro actúa se resume de la siguiente forma:

1. Se inicia un nodo maestro, y espera la conexión de los nodos esclavos.

2. Se determina la mejor forma en que los nodos deben conectarse, para reducir los tiempos de transferencia de datos entre estos.
3. Se reparte la definición del problema, entre todos los nodos esclavo, y se definen sus nodos vecinos.
4. Una vez distribuida la definición del problema, el nodo maestro tiene dos posibles opciones:
   (a) Guardar los resultados de cada iteración de la simulación, y volcar los resultados en un archivo para ser después reproducir los resultados, así mismo es posible ver la integración de todas las partículas en una sola máquina.
   (b) Avisar a los nodos de sus desconexión, para que cada nodo guarde los resultados procesados.
   (c) Vigilar y gestionar el estado de conexión entre los nodos.

Mientras que cualquier nodo esclavo se comporta de la siguiente forma:

1. Se conecta a un nodo maestro
2. Informa de los recursos que el usuario permite utilizar, y espera respuesta del nodo.
3. Recibe la definición del problema y sus nodos vecinos de conexión
4. Espera el comando para procesar las partículas
5. Procesa las partículas
6. Envia y recibe resultados de sus nodos vecinos
7. Se repite el paso 4, hasta que se indique lo contrario.

El proceso de señalización y el envio de información que permite que varias máquinas se conecten en la red, se realiza mediante el uso de una herramienta de acceso público, Peer.js (39), esta herramienta permite mostrar los resultados de la simulación en el navegador, y debido a la modularización del sistema, es posible intercambiar la herramienta para mostrar los resultados.

El modo en que el nodo maestro vigila la conexión entre los nodos, se realiza mediante la medición del tiempo que tarda en enviar y recibir información entre los nodos conectados, y dependiendo de las condiciones de la red este determinara si vale la pena cambiar el nodo vecino de conexión, con las implicaciones que tiene realizar estos cambios, ya que se necesitara enviar las partículas al nuevo nodo de conexión, mientras la simulación esta detenida.

Si por el contrario, se deja de lado el nodo maestro al no gestionar la conexión entre los nodos, cada nodo sera el encargado de negociar con sus nodos vecinos las partículas que puede enviar o recibir, con el objetivo de mantener el mayor número de partículas en la memoria de las tarjetas gráficas, sin perder o modificar las conexiones iniciales de los nodos vecinos.

4.6.2.3 Tercera fase de Conexión

La visualización de los resultados como tal no cuenta como una fase de conexión, sin embargo es considerada como una fase ya que modifica en envío de datos a través de la conexión, al modificar el número de partículas que son enviados, debido a que el usuario puede solicitar que los resultados de cada iteración sean mostrados, en lugar de solo compartir las partículas de las fronteras, modificando los tiempos de envío considerablemente.

Esta visualización de las partículas junto con las propiedades que estas poseen presentan por sí mismo una nueva problemática, es por esta razón que se utiliza una herramienta encargada para la visualización de las partículas, reduciendo los tiempos de implementación, la herramienta utilizada fue Tree.js (40), que por su fácilidad de implementación permite especificar de manera intuitiva la posición de las partículas y las propiedades que se deben mostrar, sin mencionar que esta herramienta es compatible para la mayoría de navegadores usados hoy en día, y sigue constantemente mantenida por una comunidad libre que propone, usa y reporta errores.

Así mismo una de las ventajas del uso de la API de three.js, es la posibilidad de la utilización de shaders con el objetivo de resaltar y ocultar zonas de interés. Su simple manejo de geometrías y su fácil portabilidad permite realizar la visualización de los resultados de la simulación en cualquier máquina o dispositivo móvil que tenga la capacidad WebGL.

Chapter 5

Resultados

5.1 Eficiencia del uso de las tarjetas Gráficas

El uso de las Tarjetas Gráficas de Propósito General (GPGPU) está limitado por la memoria que poseen actualmente (Apéndice A), este es un problema que si bien no afecta los resultados en la simulación, incrementa los tiempos necesarios para mostrar los resultados de esta, el uso del método SPH utilizado para simular el flujo sanguíneo, requiere de una gran cantidad de partículas para representar de manera correcta el fluido, lo que implica que si se desea una representación que refleje la realidad del movimiento del flujo sanguíneo, además de las funciones gobernantes definidas por el método, se necesita de una cantidad significativa de partículas para mostrar con precisión el movimiento de este, de ahí que sea necesario contar con mas tarjetas gráficas que cooperen entre si para acelerar los tiempos de cálculo, de lo contrario se requerirá procesar un grupo de partículas a la vez, pasar los resultados a la memoria de la CPU, procesar otro nuevo conjunto de partículas, y continuar hasta terminar de procesar todas las partículas para cada iteración, lo que significa un cuello de botella en la simulación, debido a la cantidad de memoria que debe ser enviada hacia y desde la memoria de la GPU, lo que significa que parte del tiempo que se gana de la por parte de paralelización de los cálculos realizados, sea desperdiciada por el envío de las partículas entre la tarjeta gráfica y la memoria utilizada por la CPU.

Una propuesta para medir la eficiencia del programa, utilizando múltiples procesadores es haciendo uso de la propuesta de ley de Amdahl (41):

$$S_{\text{latencia}}(s) = \frac{1}{(1-p) + \frac{p}{s}} \tag{5.1}$$

Donde:

- p Es la fracción del programa que puede ser paralelizado
- s Es el número de procesadores utilizados a procesar la fracción del programa

Sin embargo esta métrica hace de la suposición que el número de instrucciones se realizan a la misma velocidad tanto en paralelo como secuencialmente, ya que fue pensada para medir la eficiencia en multiprocesadores, esta suposición no refleja la realidad cuando se hace uso de tarjetas gráficas, ya que la velocidad de procesamiento realizada por el CPU es mayor a la velocidad del procesamiento del mismos número de instrucciones en la GPU, modificando esta métrica y considerando los tiempos de procesamiento de la CPU y la GPU, la propuesta de métrica queda como:

$$T_{apx}(N) = C(N)K_{\text{CPU}} + G(N)K_{\text{GPU}} \tag{5.2}$$

- $T_{apx}()$ Indica el tiempo aproximado que tomará realizar N operaciones, repartidas en la CPU y la GPU
- N Indica el número de partículas a ser procesadas
- $C()$ Indica el número de operaciones realizados por la CPU
- $G()$ Indica el número de operaciones realizados por la GPU
- K_{CPU} Es el tiempo promedio de ejecución de una operación para la CPU
- K_{GPU} Es el tiempo promedio de ejecución de una operación para la GPU

Donde las funciones $C()$ y $G()$ considerada las operaciones definidas por la notación $\mathcal{O}()$ para el programa evaluado, de esta forma la métrica considera el número de partículas, reflejando de manera mas realista la medición de tiempos promedio, de esta forma es posible medir el tiempo real utilizado por el uso de las tarjetas gráficas en el peor de los escenarios. Para complementar esta métrica es necesario considerar el tiempo de envío por parte de los datos usados, considerando la transferencia de información desde la memoria del host hacia la memoria de la GPU y de regreso:

$$T_{apx}(N) = C(N)K_{\text{CPU}} + G(N)K_{\text{GPU}} + T_{\text{HD}}(N) + T_{\text{DH}}(N) \tag{5.3}$$

- $T_{HD}(N)$ Indica el tiempo promedio transferencia de la CPU a la GPU
- $T_{DH}(N)$ Indica el tiempo promedio transferencia de la CPU a la GPU

La métrica anterior sera válida solo si todas las partículas procesadas pueden ser enviadas y procesadas por la GPU, adecuando las limitaciones de la memoria en las tarjetas gráficas agregando un nuevo termino para definir el máximo de memoria que puede ser manejado por la GPU, la métrica queda como:

$$T_{apx}(N, N_{GPU}) = C(N)K_{CPU} + \lceil(\frac{N}{N_{GPU}})\rceil(G(N_{GPU})K_{GPU} + T(N_{GPU})) \tag{5.4}$$

$$T(N_{GPU}) = T_{HD}(N_{GPU}) + T_{DH}(N_{GPU})$$

Donde los nuevos términos agregados definen:

N_{GPU}	Indica el máximo de partículas que puede manejar la tarjeta gráfica
$\lceil(\frac{N}{N_{GPU}})\rceil$	Indica el número de veces que se realizaran las transferencias para terminar de procesar todas las partículas
$T(N_{GPU})$	Indica el tiempo que toma la transferencia de N_{GPU} partículas

La descripción de la métrica 5.4, podemos simplificarla como:

$$T_{apx}(N) = C(N)K_{CPU} + TC_{GPU}(N) \tag{5.5}$$

Donde TC_{GPU} representa los tiempos que involucran el uso de una tarjeta gráfica:

$$TC_{GPU}(N) = \lceil(\frac{N}{N_{GPU}})\rceil(G(N_{GPU})K_{GPU} + T(N_{GPU}))$$

Agregando a esta métrica la implementación del uso de varias tarjetas gráficas el termino TC_{GPU} requiere adecuar los tiempos que cada una de estas toma en realizar las transferencias de datos y el procesamiento de estos:

$$TC_{GPU}(N) = \sum_{i}^{Num_{GPU}} \lceil(\frac{N}{N_{GPU}(i)})\rceil(G(N_{GPU})K_{GPU}(i) + T(N_{GPU}, i))$$

Donde N_{GPU}, K_{GPU} y $T()$ ya no representan valores constantes, ahora son funciones que dependen de la tarjeta gráfica i que definen sus propios tiempos de procesamiento y los tiempos de envío que varían según el modelo de cada tarjeta gráfica.

Utilizando esta métrica con los valores utilizados para simular un flujo de prueba, comparado con varios modelos de tarjeta la eficiencia para el mismo fluido, con los mismos parámetros con distinto número de partículas, los valores teóricos obtenidos son los siguientes:

Número de partículas	Tiempo estimado (segundos) en:			
	GeForce 910M	GeForce GT 640M	GeForce GTX 860M	Quadro M5000M
1000	0.0969	0.0952	0.0470	0.0332
2000	0.1307	0.0337	0.01216	0.0052
4000	0.1983	0.06758	0.02432	0.0081
8000	0.3967	0.1351	0.04865	0.0142
16000	0.7304	0.2703	0.0973	0.0285
32000	1.4623	0.5406	0.1946	0.0970
64000	2.866	1.0814	0.3892	0.1941

Table 5.1: Cálculo de los tiempos teoricos para distintas tarjetas gráficas

Los resultados muestran que el uso de tarjetas de modelo reciente permiten manejar una mayor cantidad de datos en un menor tiempo, sin embargo no siempre es posible contar con los recursos ideales, por lo que la propuesta de un sistema distribuido permite reutilizar los recursos dejados atrás, que se encuentran a nuestra disposición, así mismo estos resultados solo muestran los tiempos en una sola tarjeta y no toman en consideración los tiempos para mostrar o almacenar los resultados.

Se realizo la simulación de un fluido simple sin restricciones de frontera especiales, encerrando el fluido en un volumen cerrado, para medir los tiempos promedios que tardaba en realizar la simulación con diferente número de partículas, con el objetivo de comparar los tiempos calculados y los obtenidos en la realidad.

Los resultados muestran que el uso de la métrica propuesta miden con cierto grado de precisión los tiempos esperados, los resultados mostrados son obtenidos de un promedio de 100 mediciones, considerando únicamente los tiempos que se tarda en realizar los cálculos sin mostrarlos o almacenarlos en la computadora.

5.2 Sistema Distribuido

De manera general si se desea reducir los tiempos que toma los procesos para la solución de cualquier problema, se pueden tomar los siguientes acciones:

1. Tomar un enfoque diferente para solucionar la problemática con algoritmos mas

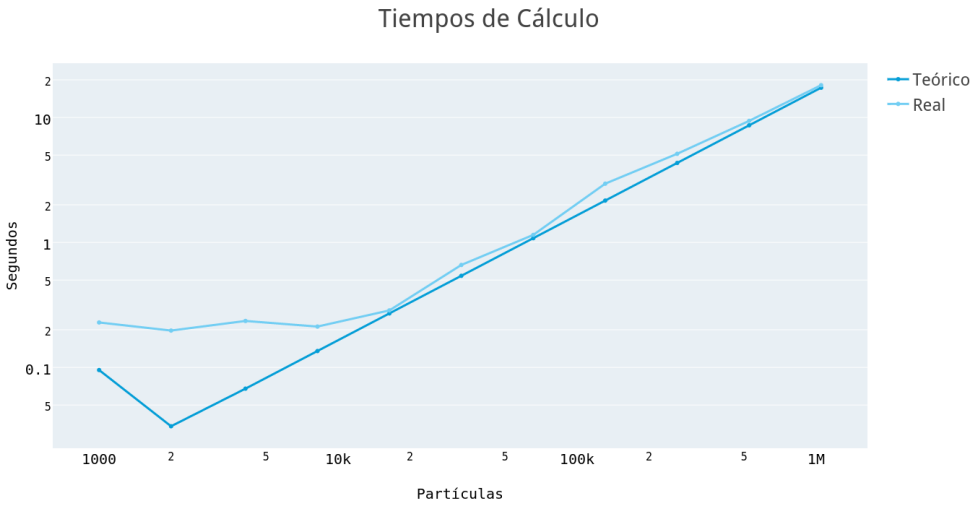

Figure 5.1: Comparación del tiempo teórico y real usando una tarjeta gráfica NVIDIA 640M

eficientes.

2. Realizar los mismos procesos con los mismos algoritmos implementados en computadoras donde se realizan estos procesos mas rápidamente.

La propuesta del uso de un sistema distribuido hace la mezcla de estos dos enfoques, con el propósito de reducir los tiempos de cálculo de la simulación, los tiempos de transferencia de información entre el host y la tarjeta gráfica son reducidos, esto se logra al distribuir los procesos y los datos en varias tarjetas gráficas manteniendo el mayor tiempo posible los datos a ser procesados dentro de cada una de estas, transfiriendo una menor cantidad de partículas entre los nodos vecinos al limitar la transferencia de las partículas, al solo enviar o recibir las que se encuentran en la frontera, reduciendo de manera significativa los tiempos de transferencia de datos entre el host y la tarjeta gráfica, sin embargo no todo es ganancia ya que al hacer el sistema distribuido se agregan nuevos tiempos al enviar y recibir las partículas entre los distintos nodos que pueden o no estar en la misma red.

La forma en que se propone medir la eficiencia de integrar este sistema distribuido es mediante la utilización de la métrica propuesta 5.6, al agregar el termino referido a los tiempos de transferencia entre los nodos, quedando la métrica de la siguiente forma:

$$T_{apx}(N) = C(N)K_{CPU} + TC_{GPU}(N) + TD(N) \tag{5.6}$$

Donde el nuevo término TD, define los tiempos de transferencia de todos los nodos, para calcular una sola iteración:

$$TD = \sum_{i=1}^{N_{GPU}} \sum_{j=1}^{N_{GPU}} TR(i,j)KR(i)$$

Donde $TR(i,j)$ es el tiempo de transferencia que se realizado del nodo i al nodo j, considerando las partículas a ser enviadas y el tiempo $KR(i)$ de transferencia. El número de partículas a ser enviadas y recibidas dependerá del problema que se este resolviendo se vera afectada el número de partículas y por lo tanto el tiempo de transferencia entre los nodos. Dejando de lado las diferencias entre las tarjetas y suponiendo se tienen dos máquinas que comparten recursos entre si las cuales son idénticas, la visualización para la distribución de las partículas en estas dos tarjetas gráficas para su interacción se muestra en la siguiente figura 5.2

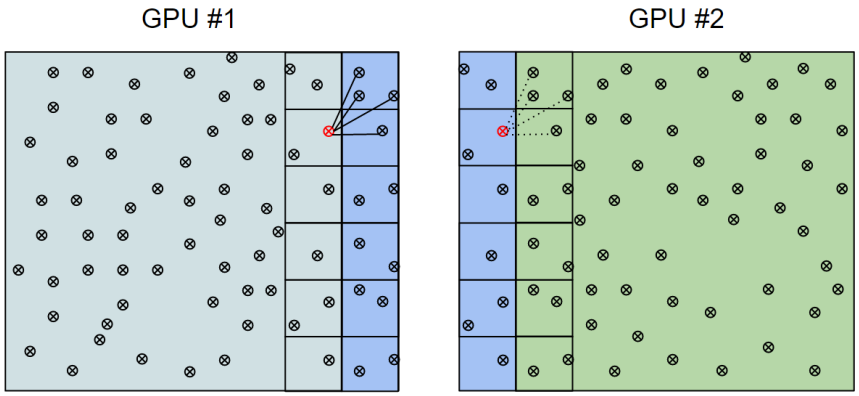

Figure 5.2: Representación de dos tarjetas compartiendo recursos

Esta solución implementada requiere de la repetición de las partículas que posiblemente interactúan con la tarjeta adyacente, estas partículas no necesitan ser procesadas, solo requieren estar presentes para la evaluación de las partículas adyacentes, por lo que además de los nuevos tiempos de transferencia, debemos considerar memoria extra para almacenar estas partículas clonadas de la frontera, lo que agrega una limitante para la memoria donde residen las partículas.

Debido a la forma en que se procesan las partículas no es necesario la repetición de todas las partículas en todas las fronteras, esto se explica con mas detalle en el capítulo 3, resumiendo esta idea, solo se requiere de la mitad de las partículas a sean enviadas, ya que se espera que sean procesadas las partículas no enviadas, por partículas adyacentes, que no se encuentran en la frontera, tal como se muestra en la figura 2.1.

Esta reducción en las partículas a ser enviadas no esta exenta de problemas, esta reducción en el envío de partículas dependerá de la problemática a resolver, en particular para el caso de estudio presentado, se espera que la transferencia de las partículas clonadas de una tarjeta mantenga la misma cantidad de partículas, y dependiendo

del movimiento del flujo, este puede provocar un pequeño aumento o descenso de las partículas clonadas. Un escenario en el que se puede ver que una sobrecargada por las partículas a procesar es cuando el flujo es acumulado en una región.

Tomando los resultados obtenidos de la tabla 5.1 algunos valores esperados para la conexión de dos máquinas con las mismas características son las siguientes, se debe notar que si todas las partículas pueden ser procesadas por una sola máquina, no se requerirá apoyarse de la segunda tarjeta gráfica, por el contrario si no le es posible procesar todas las partículas a la vez, se reparte equitativamente la cantidad de partículas, por lo que la carga de trabajo se espera sea equitativa entre los nodos conectados, pero no en todos, solo se ocuparan los nodos necesarios, de lo contrario se aumentaran los tiempos de cálculo debido a la transferencia de datos entre los nodos.

Número de partículas	Tiempo estimado haciendo uso de:			
	GeForce 910M	GeForce GT 640M	GeForce GTX	Quadro M5000M
1000	0.0969	0.0952	0.0471	0.0032
2000	0.1307	0.0337	0.0121	0.0051
4000	0.1983	0.0675	0.0243	0.0081
8000	0.3967	0.1351	0.0486	0.0142
16000	0.7304	0.2703	0.0973	0.0285
32000	1.1310	0.5406	0.1946	0.0973
64000	1.8526	1.0814	0.3892	0.1941

Table 5.2: Resumen del calculo de los tiempos promedios usando una sola tarjeta gráfica para distintos modelos

La tabla anterior solo hace uso de dos máquina con características similares, pero en la realidad puede ser cualquier número de máquinas con características distintas, el objetivo de la tabla anterior es mostrar que la cooperación de varias máquinas bajo condiciones normales de red, es decir al menos la mitad de los paquetes enviados por la red local sean debidos por la simulación, y los tiempos de espera, entre dos redes sea de menos 0.1 segundos por cada 5000 partículas, lo que muestra que sera mejor la utilización de un sistema distribuido, en lugar de solo usar una sola máquina con mejores especificaciones. Estos resultados teóricos muestran que cuando solo se envía el 10 por ciento de la simulación, que conforma la frontera del problema, entre dos

máquinas, se logrará un mejor desempeño, a si se realizara toda la simulación en una sola máquina procesando grupos de partículas a la vez.

Los resultados mostrados en la siguiente gráfica, presentan el mejor tiempo ya sea usando una sola tarjeta gráfica o utilizando ambas tarjetas con las mismas características, en caso que no sea posible procesar todas las partículas a la vez, si por el contrario no puede procesar todas las partículas en una sola tarjeta gráfica, el problema es dividido en ambas tarjetas. La estimación de los tiempos se hace con la métrica antes descrita 5.6, la cual muestra que será mas rápido el procesamiento del mismo problema que haciendo uso de dos tarjetas gráficas con las mismas características que haciendo uso de una tarjeta de mejor características, e inclusive esta casi a la par con una tarjeta de mejor características como es ek uso de una Quadro M5000M.

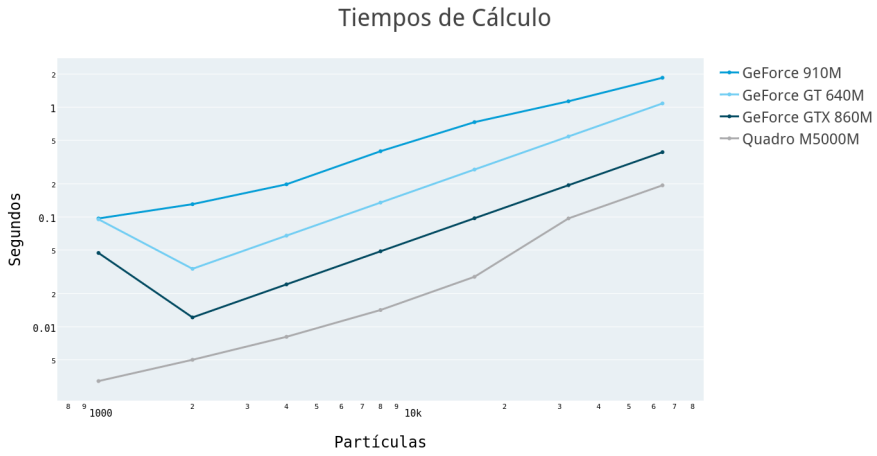

Figure 5.3: Tiempos teóricos de dos tarjetas con las mismas características para el cálculo de la simulación

La solución más simple para evitar problemas en la repartición de las partículas es mediante la definición de un máximo de memoria para las partículas repetidas, de no caber todas las partículas recibidas se procesa un bloque y después otro hasta procesar todas, transfiriendo los grupos de la memoria de la CPU a la memoria de la GPU, esta solución desperdicia tiempo cálculo, debido a las transferencias de datos, sin embargo se espera que el número de partículas de los sectores de la frontera no sea mayor que las partículas que esta procesando la tarjeta actualmente. Otra forma con la que se puede solucionar este problema en la repartición de las partículas de frontera es mediante el movimiento de las fronteras, transfiriendo los sectores que son considerados frontera, a los nodos adyacentes, de tal modo que todas las partículas de los sectores sean calculados a la vez, esto se puede lograr negociando con los nodos vecinos los sectores que serán transferidos, considerando el número de partículas que tienen los sectores y el número de partículas que serán los nuevos sectores frontera, de esta forma las fronteras entre los nodos se moverán dependiendo de la definición del problema.

5.2.1 Distribución de Sectores Frontera

El algoritmo implementado para maximizar el número de partículas en las tarjetas gráficas, y el movimiento de las fronteras considera:

1. El número de partículas extra que puede procesar la tarjeta gráfica
2. El número de sectores que considera frontera, los cuales serán enviados a otros nodos, de estos se consideran:
   (a) El número de partículas que contiene cada sector.
   (b) La ubicación respecto al nodo al que sera enviado.
3. El tiempo promedio de transferencia entre los nodos.

Por lo que en esencia el algoritmo maximiza el número de partículas que cada tarjeta gráfica puede procesar sin exceder el limite de memoria que cada tarjeta gráfica posee, minimizando el número de transferencias entre los nodos vecinos, considerando los tiempos de transferencia entre una máquina y otra. Esta medición entre los tiempos de transferencia de una máquina a otra son estimaciones que se realizan cuando se envían y reciben los datos de una máquina a otra, considerando que la red en la que se puede implementar el sistema distribuido no es exclusiva para las máquinas que procesan la simulación y pueden aumentar o disminuir la velocidad de transferencia en cualquier momento.

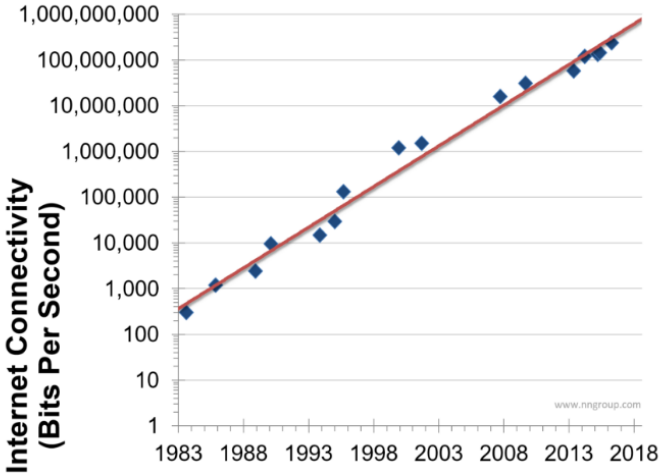

Figure 5.4: Evolución del ancho de banda en la Internet desde 1963 a la fecha.

Esta propuesta que hace uso de la Internet como red de conexión, en lugar de una red dedicada es un intento para reducir los costos de manutención y creación de una red con las características ideales, hace 30 años o menos no hubiera sido factible la utilización de la Internet como medio de conexión entre las máquinas debido al ancho

de banda que rondaba entre los 90 KB/s y los 128 kB/s de transferencia 5.4, el aumento del ancho de banda ha permitido contemplar la utilización de esta red sin la necesidad de la creación de una red exclusiva para las máquinas que realizan la simulación, de seguir con el patrón de crecimiento, que se ha tenido hasta el momento, en algunos años sera mejor tener un sistema distribuido con estas características, que las redes locales que se tienen actualmente, permitiendo ahorrar costos extra, al permitir agrupar recursos de varias partes, permitiendo incrementar el poder computacional, una alternativa que reduce costos a expensas de la seguridad de la información que será enviada en la red pública.

De igual forma que con el ancho de banda, que ha aumentado casi de manera exponencial en las últimas décadas (42), también ha aumentado la capacidad de cómputo por parte de los procesadores (43), lo que significa que ha permitido condiciones ideales para la consideración de un sistema distribuido, haciendo uso de la Internet con las máquinas actuales, no solo en máquinas de escritorio sino también en dispositivos móviles como dispositivos de visualización, como lo hace la presente implementación.

5.3 Interfaz del Simulador

Para el simulador de fluidos implementado, se desarrollo una interfaz amigable que permite observar los resultados realizados por el simulador, de igual forma se desarrollo una interfaz que permita a los usuarios que desean compartir recursos computaciones conectarse a un nodo maestro para la gestión de los recursos.

Las interfaces de conexión entre el nodo maestro y los clientes así como el visualizador de resultados fueron desarrollados en HTML, debido a su fácil portabilidad y uso.

Además de las interfaces realizadas para mostrar los resultados que son calculados en tiempo real, y del visualizador para poder mostrar los resultados calculados, se creo una interfaz que permite conectar cualquier dispositivo con capacidad de renderizado WebGL, que se encuentre conectado a Internet, mostrar los resultados que están siendo procesados por el sistema distribuido en tiempo real.

Estas interfaces hacen uso de un Websocket para comunicarse con el adaptador que se comunica con el simulador, de igual forma que con las interfaces implementadas este visualizador se desarrollo en HTML y javascript, ya que permite una fácil manipulación de los datos y permite su uso sin requerir de la instalación de algún programa en especial, para poder mostrar los resultados, ejemplo de esto es el uso de la API Tree.js, que permite mostrar las posiciones de las partículas como una nube de puntos para su visualización, y con la ayuda de javascript es posible modificar la forma en que es presentada la nube de puntos, ya sea de manera normal en un espacio cartesiano XYZ, o bien con la ayuda de un shaders sencillo, es posible distorsionar el resultado final de la nube de puntos, para de esta forma mostrar la nube de puntos en un visualizador 3D, como lo son los dispositivos Oculus Rift o Gear VR (44), o de manera mas sencilla con ayuda de los Google Cardboard (45).

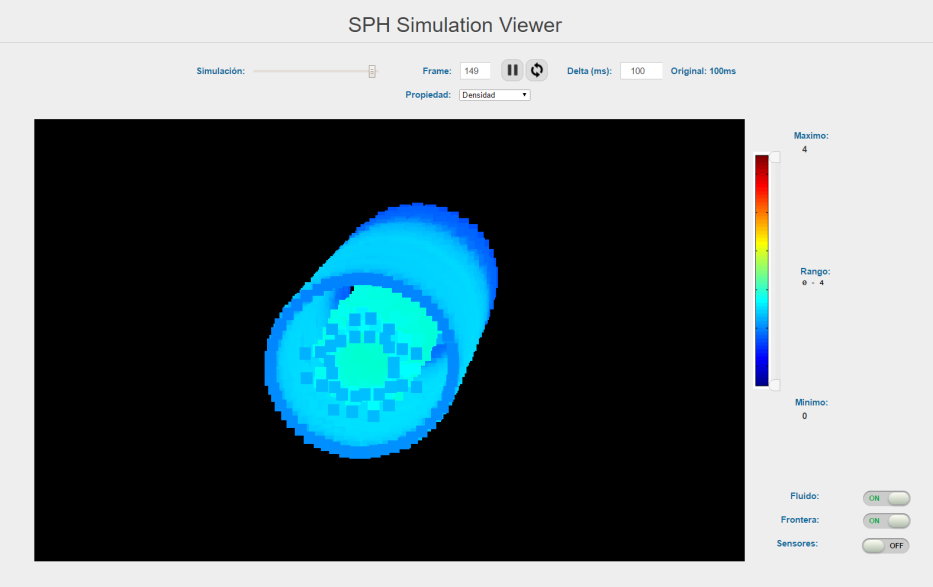

Figure 5.5: Visualización de un segmento de arteria, mostrando la pared arterial y el fluido de la sangre

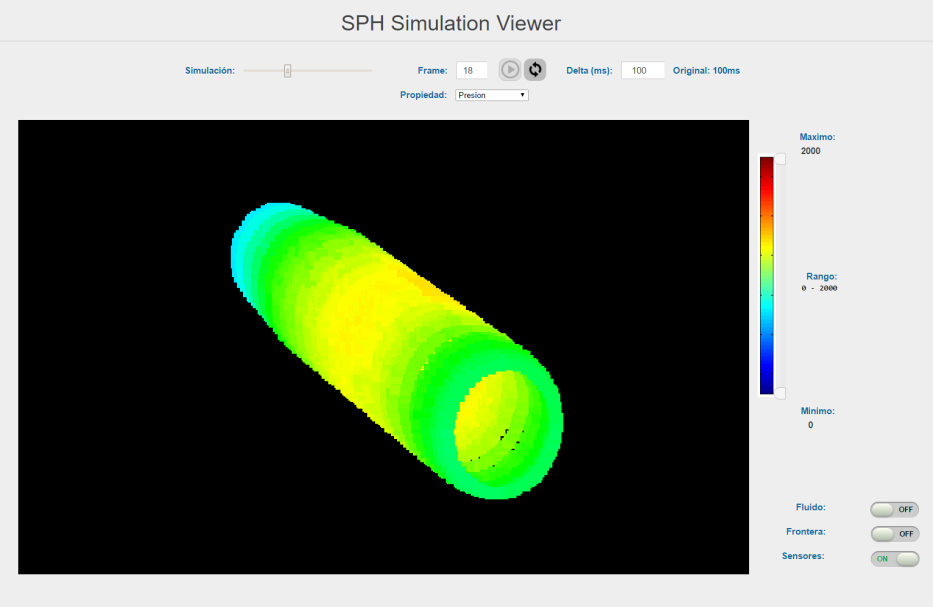

Figure 5.6: Visualización de la presión que se ejerce sobre la pared arterial, debido al flujo sanguíneo

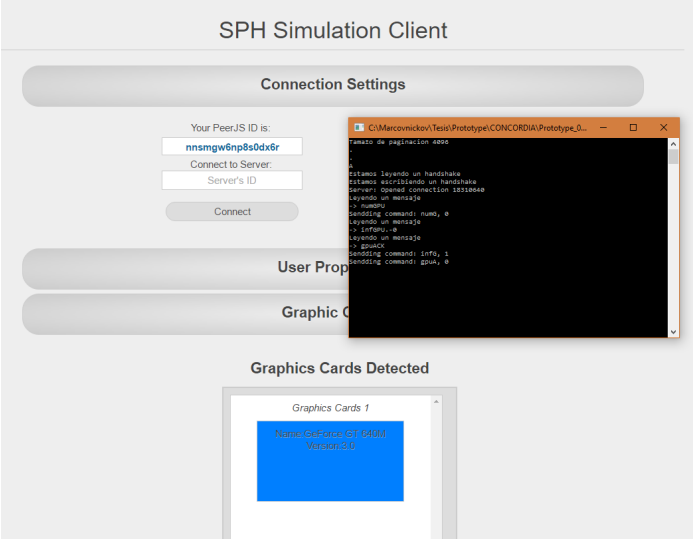

Figure 5.7: Visualizador del Cliente, mostrando la respuesta de la página con el adaptador para recibir las características de la tarjeta gráfica

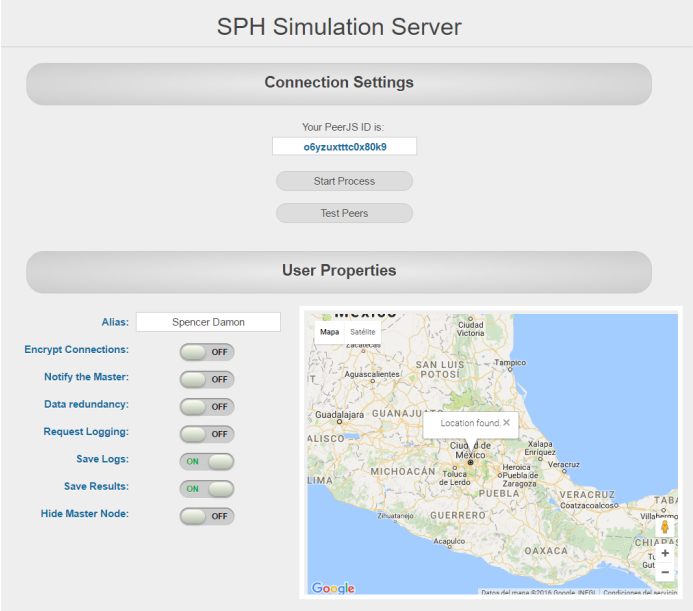

Figure 5.8: Visualizador del Nodo Maestro

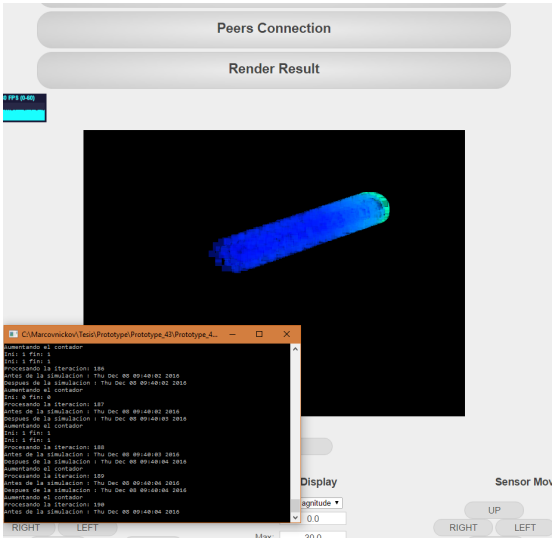

Figure 5.9: Visualización de una iteración de la simulación, haciendo uso de una sola tarjeta gráfica

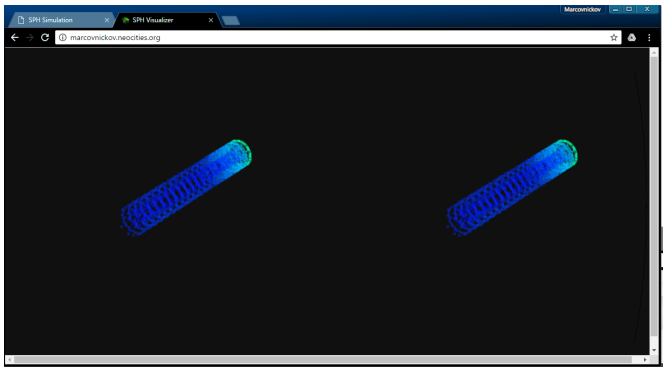

Figure 5.10: Visualización de las partículas como una nube de puntos, renderizada para ser vista por los Oculus Rift o Google Cardboard

Chapter 6

Conclusiones

La presente trabajo tiene como objetivo, el desarrollo de una simulación de flujos y de manera particular el fluido biológico de la sangre, mediante el método de SPH esta implementación ha mostrado ser una herramienta que permite estimar con cierto grado de precisión el comportamiento de la sangre a través de las arterias, mostrando la interacción de la sangre y de las paredes arterias, así como la interacción con objetos ajenos a esta, como es la introducción de un catéter ó la colocación de un stent, mostrando los posibles comportamientos de las paredes arteriales y la presión de la sangre cuando estos interactúan, así mismo ha mostrado resultados aceptables para la simulación de un flujo homogéneo, a través de un tubo, esta prueba de simulación permitió evaluar la evaluación del método SPH, sin embargo la prueba realizada no es comparable para validar completamente el comportamiento de la sangre.

Las características que pueden mejorarse en un futuro es el manejo de errores por parte del sistema distribuido, ya que de momento solo se detectan las fallas en la red, pero no se toman acciones automáticamente, y es el usuario quien debe manejar estos errores, así mismo la simplificación de la sangre da lugares a errores en la estimación del comportamiento en arterias de menor tamaño, esta simplificación de la sangre como fluido homogéneo es utilizado para imitar las características de la sangre como medio de transporte, en segmentos de arterias musculares, sin embargo las definición actual de la sangre tomada en las simulaciones no permite la simulación en segmentos de arteria mas pequeños, por lo que se deben considerar en un futuro, el fluido biológico de la sangre como un fluido compuesto, que recolecta y libera subproductos del torrente sanguíneo, sin embargo para el objetivo planteado inicialmente, la simplificación de la sangre como un fluido único, permitió estimar algunas interacciones con el fluido sanguíneo y objetos ajenos a este.

Los recursos que el sistema utiliza no esta a la para por mostrar resultados, que permita sustituir mediciones reales, sin embargo con ayuda del sistema distribuido desarrollado, la conexión a través con dispositivos dedicados a la entrada y/o salida de información de parámetros biológicos, permitirá agregar mayor realismo a las simulaciones realizadas, ya sea de dispositivos dedicados o dispositivos de propósito general con menores características como es la Arduino y la Raspberry Pi, que permitirán crear

interfaces con pacientes reales, en futuras implementaciones.

Por parte de los datos biológicos introducidos, para simular el flujo sanguíneo, fueron una estimación de los posibles parámetros que mayormente afectan el comportamiento de la sangre, sin embargo estos no son los únicos factores que modifican el comportamiento de la sangre, por lo que es necesario agregar nuevas ecuaciones gobernantes que modelen nuevas para el comportamiento de la sangre así como los parámetros necesarios para mejorar la simulación, un ejemplo de estos es el nivel PH en la sangre, como factor que puede afectar la temperatura de la sangre, así como las posibles reacciones químicas que suceden a nivel celular en cada una de los elementos que componen la sangre.

El presente trabajo alcanzo el objetivo planteado de desarrollar un simulador de flujos, con el caso particular del fluido biológico de la sangre, validando la simulación con fluidos simples, como se detalla en el capítulo5, además de este objetivo también se logro implementar un sistema distribuido, permite la utilización de recursos en máquinas que no necesariamente se encuentran en la misma red, permitiendo aumentar la capacidad de cómputo, así mismo este sistema distribuido permite la visualización de datos en cualquier dispositivo que se encuentre conectado a la Internet, con capacidad de renderizado WebGL, una característica que no se planeo, pero se logro gracias a la modularización del sistema desarrollado.

Apéndices

Appendix A

Evolución de las tarjetas CUDA

En el presente apéndice se muestra la evolución de las tarjetas gráficas NVIDIA, mostrando solo las tarjetas gráficas con capacidad CUDA las cuales pueden ser utilizadas como nodos en la cooperación para la resolución de la simulación, comparando solo las tarjetas gráficas a partir de la arquitectura Fermi 2.0, las cuales permiten el uso de todas las operaciones implementadas en la simulación, como son operaciones atómicas, manejo de operaciones flotantes de doble precisión, manejo de texturas, entre otras.

La lista de posibles tarjetas gráficas que son candidatas para su posibles conexión se presentan a continuación [1], clasificándolas por primero por modelo, nombre y arquitectura, para que sea mas fácil su ubicación. Cabe mencionar que los datos presentados son obtenidos de la pagina de NVIDIA (46) y resumen las propiedades deseadas para comparar su posible uso en la en su uso para su posible uso en la cooperación para procesar la simulación.

[1] Son el listado completo de tarjetas hasta la fecha que teóricamente pueden ser usadas en la simulación, no significa que se hallan implementado o probado en todas estas

Modelo	Nombre	Arquitectura	Reloj Núcleo (MHz)	Reloj Memoria (MHz)	Bus de Memoria	Tipo de Memoria	Máximo de Memoria (MB)
GeForce	610M	Fermi	672	1800	64	DDR3	1024 / 2048
	710M	Fermi	775	1800	64	DDR3	2048
	820M	Fermi	775	1800	64	DDR3	2048
	825M	Kepler	850	1800	64	DDR3	2048
	830M	Maxwell	1029	1800	64	DDR3	2048
	840M	Maxwell	1029	2000	64	DDR3	4096
	845M	Maxwell	1071	2000 - 5000	64/128	DDR3/GDDR5	2048
	910M	Kepler	641	2000	64	DDR3	1024
	920M	Kepler	954	1800	64	DDR3	4096
	920MX	Maxwell	954	1800	64	DDR3	4096
	930M	Maxwell	928	1800	64	DDR3	2048
	930MX	Maxwell	928	1800	64	DDR3	2048
	940M	Maxwell	1072	2000	64	DDR3 / GDDR5	4096
	940MX	Maxwell	1122	1800	64	DDR3/GDDR5	4096
	945M	Maxwell	928	2000	128	DDR3	4096
	GT 520M	Fermi	740 / 600	800 / 900	64 / 128	DDR3 / GDDR5	1536
	GT 520MX	Fermi	900	900	64	DDR3	1536
	GT 525M	Fermi	600	900	128	GDDR5, DDR3	1536
	GT 540M	Ferm	672	900	128	GDDR5, DDR3	1536
	GT 550M	Fermi	740	900	128	DDR3/GDDR5	1536

Table A.1: Resumen de Tarjetas Gráficas I

Modelo	Nombre	Arquitectura	Reloj Núcleo (MHz)	Reloj Memoria (MHz)	Bus de Memoria	Tipo de Memoria	Máximo de Memoria (MB)
GeForce	GT 555M	Fermi	525	785-900	128/192	DDR3/GDDR5	3072
	GT 620M	Fermi	625	1800	64 / 128	DDR3	1024
	GT 625M	Fermi	625	1800	64	DDR3	1024
	GT 630M	Fermi	672	1800	128	DDR3	1024
	GT 635M	Fermi	660	1800	128/192	DDR3/GDDR5	2048
	GT 640M	Kepler	625	1800 - 4000	128	DDR3, GDDR5	2048
	GT 640M LE	Fermi / Kepler	500	1800 - 4000	128	DDR3, GDDR5	2048
	GT 645M	Kepler	710	1800	128	DDR3	2048
	GT 650M	Kepler	735	1800 - 4000	128	DDR3 / GDDR5	2048
	GT 720M	Fermi	625	1800 - 2000	64	DDR3	2048
	GT 730M	Kepler	725	1800 - 2000	64/128	DDR3	4096
	GT 735M	Kepler	575	1800 - 2000	64	DDR3	2048
	GT 740M	Kepler	810	1600 - 1800	64/128	DDR3	2048
	GT 745M	Kepler	837	1800	128	DDR3, GDDR5	2048
	GT 750M	Kepler	967	2000 - 5000	128	DDR3, GDDR5	4096
	GT 755M	Kepler	980	5400	128	GDDR5	4096
	GTX 1050 [2]	Pascal	1354	7000	128	GDDR5	2048
	GTX 1050 Ti [2]	Pascal	1493	7000	128	GDDR5	4096
	GTX 1060 [2]	Pascal	1506	8000	192	GDDR5	6144
	GTX 1070 [2]	Pascal	1443	8000	256	GDDR5	8192

Table A.2: Resumen de Tarjetas Gráficas II

Modelo	Nombre	Arquitectura	Reloj Núcleo (MHz)	Reloj Memoria (MHz)	Bus de Memoria	Tipo de Memoria	Máximo de Memoria (MB)
GeForce	GTX 1070 SLI [2]	Pascal	1443	8000	2x 256	GDDR5	2 x 8192
	GTX 1080 [2]	Pascal	1566	10000	256	GDDR5X	8192
	GTX 1080 SLI [2]	Pascal	1607	10000	2x 256	GDDR5	2 x 8192
	GTX 560M	Fermi	775	1250	192	GDDR5	1536
	GTX 570M	Fermi	575	1150	192	GDDR5	1536
	GTX 580M	Fermi	620	1500	256	GDDR5	2048
	GTX 660M	Kepler	835	4000	128	GDDR5	2048
	GTX 670M	Fermi	598	3000	192	GDDR5	3072
	GTX 670MX	Kepler	600	2800	192	GDDR5	2048
	GTX 675M	Fermi	620	3000	256	GDDR5	2048
	GTX 675MX	Kepler	600	3600	256	GDDR5	2048
	GTX 680M	Kepler	720	3600	256	GDDR5	4096
	GTX 680MX	Kepler	720	5000	256	GDDR5	4096
	GTX 760M	Kepler	657	4000	128	GDDR5	2048
	GTX 765M	Kepler	850	4000	128	GDDR5	2048
	GTX 770M	Kepler	811	4000	192	GDDR5	3072
	GTX 775M	Kepler	719	3600	256	GDDR5	4096
	GTX 780M	Kepler	823	5000	256	GDDR5	4096
	GTX 850M	Maxwell	876	2000 - 5000	128	DDR3 / GDDR5	4096
	GTX 860M	Maxwell	1029	5000	128	GDDR5	4096

Table A.3: Resumen de Tarjetas Gráficas III

Modelo	Nombre	Arquitectura	Reloj Núcleo (MHz)	Reloj Memoria (MHz)	Bus de Memoria	Tipo de Memoria	Máximo de Memoria (MB)
GeForce	GTX 870M	Kepler	941	5000	192	GDDR5	6144
	GTX 880M	Kepler	954	5000	256	GDDR5	8192
	GTX 880M SLI	Kepler	954	5000	2x 256	GDDR5	2x8192
	GTX 950M	Maxwell	914 - 993	1800 - 5000	128	DDR3/GDDR5	4096
	GTX 960M	Maxwell	1029	5000	128	GDDR5	4096
	GTX 965M	Maxwell	924 / 935	5000	128	GDDR5	4096
	GTX 965M SLI	Maxwell	924	5000	2x 128	GDDR5	2 x 4096
	GTX 970M	Maxwell	924	5000	192	GDDR5	6144
	GTX 970M SLI	Maxwell	924	5000	2x 192	GDDR5	2 x 6144
	GTX 980 [2]	Maxwell	1126	3500	256	GDDR5	4096
	GTX 980 SLI [2]	Maxwell	1126	3500	2x 256	GDDR5	2 x 8192
	GTX 980M	Maxwell	1038	5000	256	GDDR5	8192
	GTX 980M SLI	Maxwell	1038	5000	2x 256	GDDR5	2x 8192
	GTX 940M	Maxwell	954	5012	64	GDDR5	1024
NVS	4200M	Fermi	810	800	64	DDR3	1024
	5200M	Fermi	625	1800	64	DDR3	1024
	5400M	Fermi	660	1800	128	DDR3	2048
Quadro	1000M	Fermi	700	900	128	DDR3	2048
	2000M	Fermi	550	900	128	DDR3	2048
	3000M	Fermi	450	625	256	GDDR5	2048

Table A.4: Resumen de Tarjetas Gráficas IV

Modelo	Nombre	Arquitectura	Reloj Núcleo (MHz)	Reloj Memoria (MHz)	Bus de Memoria	Tipo de Memoria	Máximo de Memoria (MB)
Quadro	4000M	Fermi	475	1200	256	GDDR5	2048
	5010M	Fermi	450	1300	256	GDDR5 ECC	4096
	K1000M	Kepler	850	1800	128	DDR3	2048
	K1100M	Kepler	705	2800	128	GDDR5	2048
	K2000M	Kepler	745	1800	128	DDR3	2048
	K2100M	Kepler	667	3000	128	GDDR5	2048
	K3000M	Kepler	654	2800	256	GDDR5	4096
	K3100M	Kepler	706	3200	256	GDDR5	4096
	K4000M	Kepler	600	2800	256	GDDR5	4096
	K4100M	Kepler	706	3200	256	GDDR5	4096
	K5000M	Kepler	706	3000	256	GDDR5	4096
	K500M	Kepler	850	1800	64	DDR3 / GDDR5	2048
	K5100M	Kepler	771	3600	256	GDDR5	8192
	K510M	Kepler	846	2400	64	GDDR5	1024
	K610M	Kepler	954	2600	64	GDDR5	1024
	K620M	Maxwell	1029	2000	64	DDR3	2048
	M1000M	Maxwell	993	5000	128	GDDR5	4096
	M2000M	Maxwell	1038	5000	128	GDDR5	4096
	M3000M	Maxwell	1050	5000	256	GDDR5	4096
	M4000M	Maxwell	975	5012	256	GDDR5	4096

Table A.5: Resumen de Tarjetas Gráficas V

Modelo	Nombre	Arquitectura	Reloj Núcleo (MHz)	Reloj Memoria (MHz)	Bus de Memoria	Tipo de Memoria	Máximo de Memoria (MB)
Quadro	M5000M	Maxwell	962	5000	256	GDDR5	8192
	M500M	Maxwell	1029	4004	64	DDR3	2048
	M5500	Maxwell	1139	6606	256	GDDR5	8192
	M600M	Maxwell	837	5012	128	GDDR5	2048

Table A.6: Resumen de Tarjetas Gráficas VI

Appendix B

Introducción al cómputo en Paralelo

Esta breve introducción permitirá entender las ventajas y desventajas que se tienen al procesar las partículas vecinas propuestas, la anterior explicación se debe tomar como una guia rápida que no explica a detalle todas las características del uso de la API de CUDA, solo sirve como una breve introducción, ya que cada vez se agregan y desechan características nuevas, de esta guia rápida se debe rescatar que el manejo de memoria para su declaración, manejo y liberación, así como la sincronización de las funciones kernel son puntos claves para poder medir la viabilidad de implementación de cualquier programa a ser ejecutado en paralelo, ya que el cómputo en paralelo no garantiza un mejor desempeño, y dependerá de la problemática a ser resuelta, sin embargo para esta problemática en particular, que se pretende resolver es sin lugar a dudas una clara mejora respecto a solo implementar en la CPU, utilizando la documentación proporcionada por NVIDIA (47).

B.0.1 Memoria

La forma en que se define y maneja la memoria en el cómputo paralelo difiere a como se define y maneja para programas que son ejecutados secuencialmente por la CPU. La memoria del host referido a la memoria utilizada por el procesador del CPU y la memoria de la tarjeta gráfica, residen en espacios separados y no tienen interacción directa entre estas, esto es reflejado por la forma física en que son ensamblados estos elementos, cada tarjeta gráfica utilizan una memoria independiente para realizar los procesos, lo que permite que estos cálculos sean procesados independientemente de los procesos que realiza la CPU lo que es una ventaja y una desventaja si no se maneja correctamente los datos a ser ocupados.

Para poder realizar cualquier cálculo que involucre una gran cantidad de datos es necesario primero definir la memoria donde serán almacenados estos, independientemente que ya existan almacenados en la memoria del host, una vez definida la memoria a utilizar en la tarjeta gráfica es posible transferir los datos de la memoria del host a esta nueva memoria declarada, una vez terminada de transferir todos los datos necesarios realizar los cálculos, al terminar de procesar estos de ser necesario se realizará la

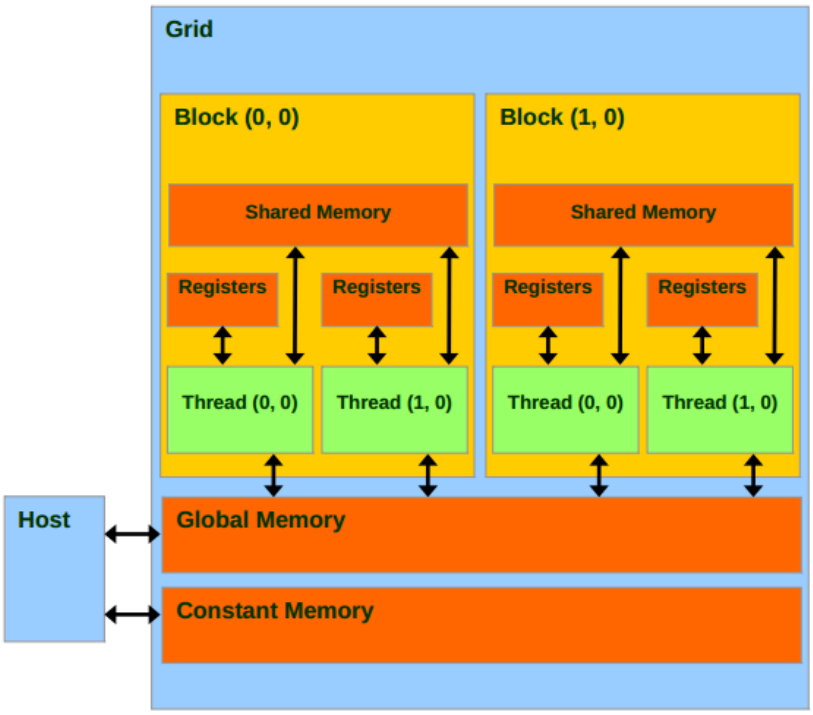

Figure B.1: Representación de la división de threads usando la API de CUDA.

transferencia de los resultados nuevamente a la memoria del host, finalmente si ya no se requiere utilizar la memoria declarada en la tarjeta gráfica es necesario liberar la memoria declarada, estos pasos describen superficialmente el ciclo de vida de la memoria en la tarjeta gráfica.

Además de la separación física entre la memoria del host y la memoria de las tarjetas gráficas, la memoria localizada en las tarjetas gráficas esta dividida con el objetivo de optimizar procesos específicos estos tipos de memoria son: memoria de constantes, memoria de registros, memoria compartida y memoria global, cada una de estas posee sus propias ventajas y desventajas de uso, siendo la memoria de los registros la mas veloz en comparación con la memoria global que es mas lenta en su lectura y escritura.

Los tipos de memoria que permite la transferencia de datos entre el host y la tarjeta gráfica son la memoria constante y la memoria global, mientras que la memoria de los registros, memoria global y memoria constante solo puede ser accedida por la tarjeta gráfica, los tiempos de acceso promedio son indicados en la tabla B.1, pero dependerá principalmente de cada una de las tarjetas gráficas.

Estos tiempos de accesos son aproximados y dependerán de las especificaciones de cada tarjeta, pero en general siguen el mismo patrón de velocidad, ya que fueron plateadas desde un inicio para optimizar procesos específicos, por lo que antes de im-

Tipo de Memoria	Tiempos de Acceso Aproximados
Memoria de los Registro	≈ 8,000 GB/s
Memoria Compartida	≈ 1,600 GB/s
Memoria Global	≈ 177 GB/s
Memoria Constante	≈ 8 GB/s solo lectura

Table B.1: Resumen de los tiempos de acceso para los diferentes tipos de memoria usados en CUDA

plementar cualquier programa haciendo uso de las tarjeta gráficas es necesario revisar la cantidad de memoria que se usará y como se pretende utilizar esta, ya que dependiendo de estas características dependerá la viabilidad o no de implementar el programa haciendo uso del cómputo en paralelo.

B.0.2 Transferencias de memoria

La transferencia de datos entre el host y la tarjeta gráfica, se realiza mediante una declaración explicita utilizando los métodos definidos por la API de CUDA, así mismo es posible el uso de variables y métodos de APIs adicionales como lo es la API de Thrust que facilitan la declaración, transferencia y liberación de memoria, el uso de estas se debe de hacer con precaución ya que si no se conoce el funcionamiento interno de estos métodos podría provocar un desempeño lento en las implementaciones realizadas, sea cual sea la forma en que se declaran las variables, una consideración que se debe tener en cuenta es la transferencia de los datos a ser utilizados, se debe esperar a que se terminen de transferir todos los datos necesarios para poder empezar a utilizar estos. Una manera en que CUDA se protege de las inconsistencias en los datos a utilizar, es mediante la sincronización explicita en la trasferencia de los datos, al impedir de manera automática la ejecución de cualquier función kernel hasta que la transferencia de memoria sea completada, esta sincronización no es obligatoria y puede ser evitada si es definido por el programador, lo que permitirá la ejecución de funciones kernels que no estén haciendo uso de la memoria que se esta transfiriendo, esta eliminación de los

mecanismos de protección dependerá del problema y la forma en que se hace solución al problema, así como de la tarjeta gráfica usada que permita evitar estos puntos de control.

B.0.3 Lanzamiento de Kernels

En CUDA una función kernel especifica el código a ser ejecutado en cada uno de los threads, la forma de paralelización que la API de CUDA que implementa en las tarjetas gráficas es una instancia Single-Program Multiple-Data (SPMD), este tipo de enfoque especifica el mismo código a ser ejecutado en cada uno de los threads, utilizando diferente datos a ser procesados en cada uno de estos.

Existen diferentes tipos de función kernel, las dos principales son las que son invocadas por el host, estas definen el número de bloques y el número de threads que serán ejecutados y las que solo pueden ser ejecutadas por la tarjeta gráfica, estos dos tipos de funciones kernel pueden ser invocadas en todas las tarjetas gráficas NVIDIA con capacidad CUDA, mas recientemente otro tipo de invocación permite lanzar funciones kernel que permitan definir nuevos bloques con nuevos threads para su uso, desde funciones kernel ejecutadas dentro de la tarjeta gráfica (Paralelismo Dinámico), esta invocación solo es posible ser implementada en tarjetas con arquitectura 3.5 o superior, las cuales no son consideradas en la implementación.

Cuando una función kernel es invocada, en cada uno de los threads solicitados se implementa el código a ser ejecutado en tiempo de ejecución, y a menos que se especifique de otra forma y sea soportado por la tarjeta gráfica, cada función kernel es ejecutada una a la vez, a esto se le conoce como sincronización implícita.

Cuando el host invoca una función kernel, definiendo el número de bloques y threads como parámetros de configuración de ejecución, estos no pueden ser modificados una vez lanzado el kernel, y estos no serán liberados hasta que cada uno de estos terminen su trabajo, de igual forma una vez que son invocadas las funciones kernel los apuntadores de memoria que son enviados como parámetros deben apuntar a espacios de memoria previamente definidos en la tarjeta gráfica, teniendo en cuenta que la función kernel no puede acceder a la memoria del host, al menos de manera óptima ya que es posible utilizar un tipo de memoria compartida que es posible accesar entre el host y la tarjeta gráfica, que no es considerada debido a sus lentos tiempos de acceso, ya que es mas lenta que la memoria global, así mismo esta memoria no permite su paginación lo que hace que los accesos a los recursos de esta memoria por parte del host también sea mas lento.

B.0.4 Sincronización

La sincronización de procesos es importante debido a que cada thread trata de manera independiente el mismo código y dependiendo de las condiciones de este y de los datos que maneja sera el tiempo en que terminara de procesar el mismo segmento de código antes o después que los otros threads, esto presenta problemas si algún cálculo requiere que todos los threads procesen ciertos segmentos de código antes de continuar, un ejemplo de esta problemática puede ser vista en la suma de vectores donde es posible lanzar un bloque con tantos threads como dimensiones tenga el vector para que cada thread se encargue de sumar los valores de una dimensión y almacenar en un espacio de memoria el valor resultante, sin embargo si se trata de realizar el producto punto de dos vectores, nuevamente podemos definir que cada thead realice la multiplicación de cada valor en cada dimensión, sin embargo se requiere que todos los threads hallan realizado su multiplicación antes de realizar la suma de cada uno de estos valores, para asegurar consistencia en los resultados se requiere de un mecanismo de sincronización, que permita esperar a todos los threads hasta que hallan realizado la multiplicación para sumar los valores, y debido a que todos tienen el mismo código, se requiere de una barrera de sincronización que espere hasta pasar cierto punto en la ejecución del código.

Dependiendo del modo en que se lancen las funciones kernel con el número de bloques y threads en estos, sera el tipo de sincronización requerida, por ejemplo si así lo permite el problema y todos los threads son ejecutados por un solo bloque podemos hacer uso de la memoria compartida para almacenar los resultados en el mismo espacio de memoria, y realizar una sincronización de bloque, lo que permitirá que cada thread espere a sus compañeros y una vez que todos pasen la barrera de sincronización, seguir la ejecución del código, por el contrario si la implementación requiere de varios bloques, se requerirá una barrera de sincronización que utilice la memoria global para determinar que todos los bloques hallan terminado de realizar sus procesos, y de esta forma poder continuar. Dependiendo del problema que se quiera resolver dependerá del número de funciones kernel que se requerían para solucionarlo, para el ejemplo del

producto punto un propuesta factible, es utilizar un kernel que se encargue de realizar la multiplicación de los valores en cada dimensión del vector y lanzar otro kernel con un solo thread encargado de sumar los valores y almacenarlos en un espacio de memoria, sin necesidad de una barrera de sincronización explicita dejando que las propias funciones se ejecuten una después de otra, liberando los recursos mas rápidamente y utilizando menos recursos a la vez.

Appendix C

Diagramas Técnicos

Los diagramas que se presentan a continuación son parte de la documentación presentada que tiene como objetivo mostrar el funcionamiento del sistema desarrollado, y esta divido en tres secciones, en la primera sección se describe el funcionamiento del sistema del simulador, en la segunda sección se detalla el funcionamiento del modulo CONCORDIA y finalmente en la tercera sección se describen las acciones implementadas en el navegador que permite la comunicación entre las máquinas, ya sea que estén conectadas en la misma red o no. Los diagramas presentados usan el Lenguaje Unificado de Modelado (UML) 2.5 (48)

C.1 Simulador SPH

La forma en que se presenta los diagramas del sistema de simulación es presentado la máquina de estado y diagrama de clases.

C.1.1 Máquina de Estados

El primer diagrama que se presenta C.1, resume el comportamiento del sistema de simulación considerando el uso del sistema distribuido Concordia, este refleja el comportamiento de la clase *SPH_Concordia.cpp*.

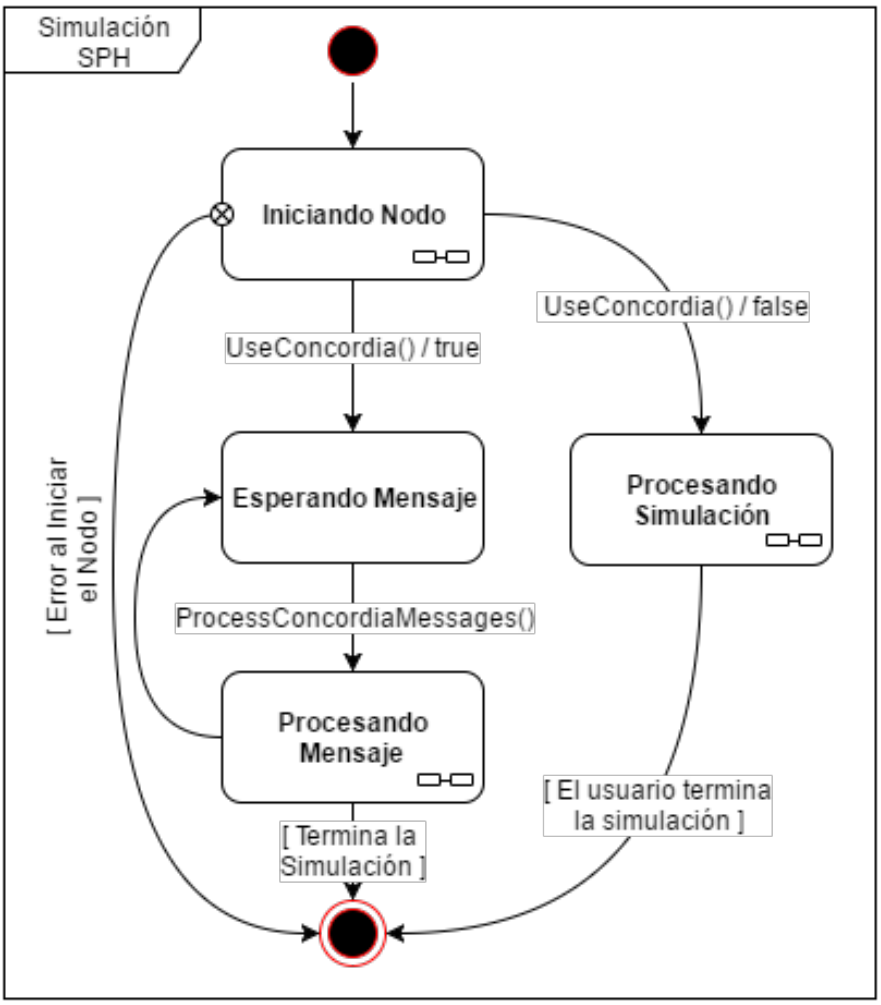

Figure C.1: Máquina de Estados del Simulador

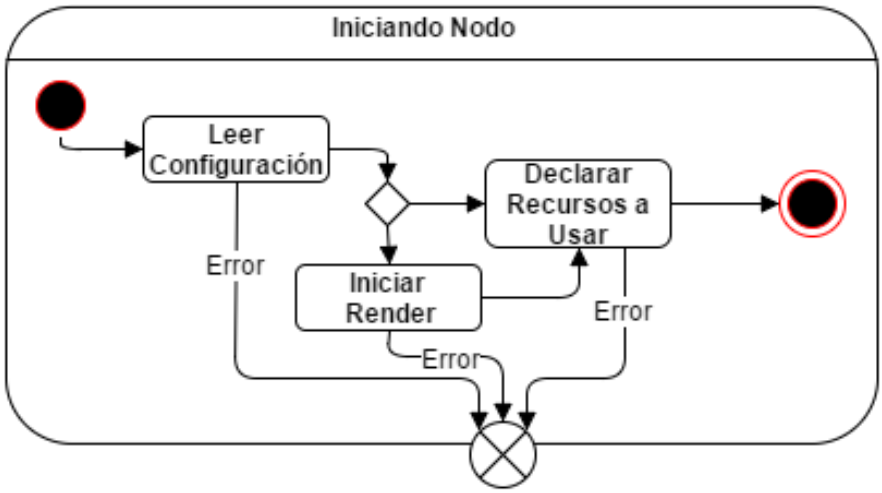

Figure C.2: Estado : Iniciando Nodo

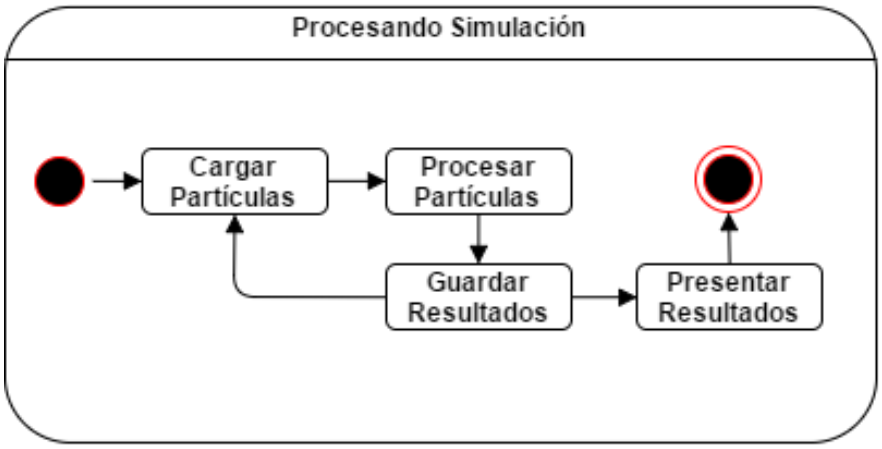

Figure C.3: Estado : Procesando Simulación

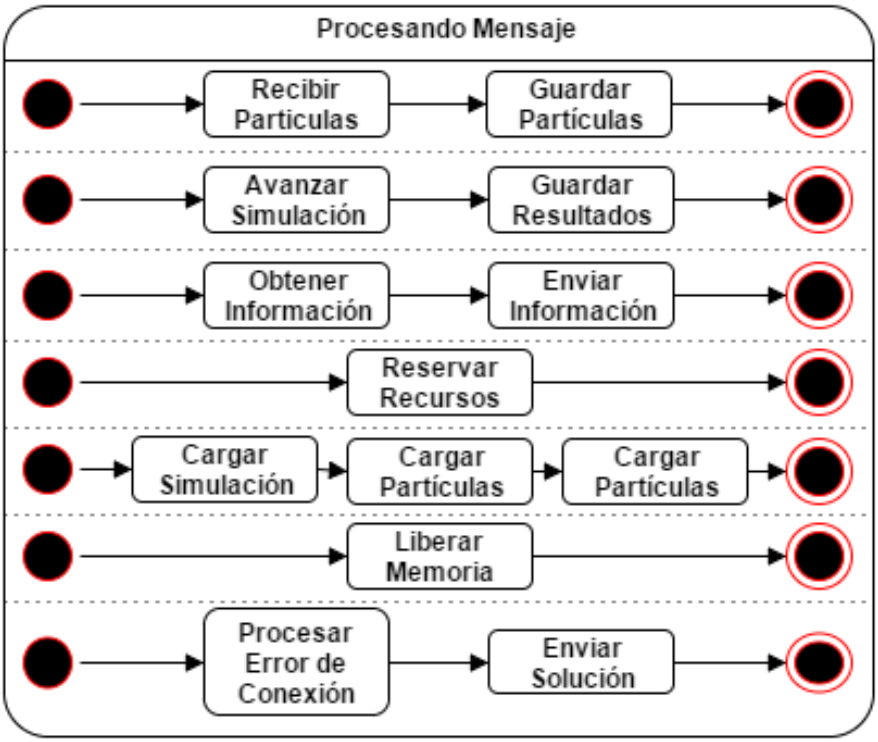

Figure C.4: Estado : Procesando Mensjae

C.1.2 Diagramas de Clases

Debido a las limitaciones de espacio, se presentan los diagramas de cada clase, mostrando en algunos casos la conexión con la clase pero no los atributos que la componen, variables y métodos.

El primer diagrama que se presenta C.5 muestra las dependencias con las clases necesarias para llevar a cabo el funcionamiento descrito por la máquina de estados C.1. A continuación se presentan los atributos de cada clase involucrada.

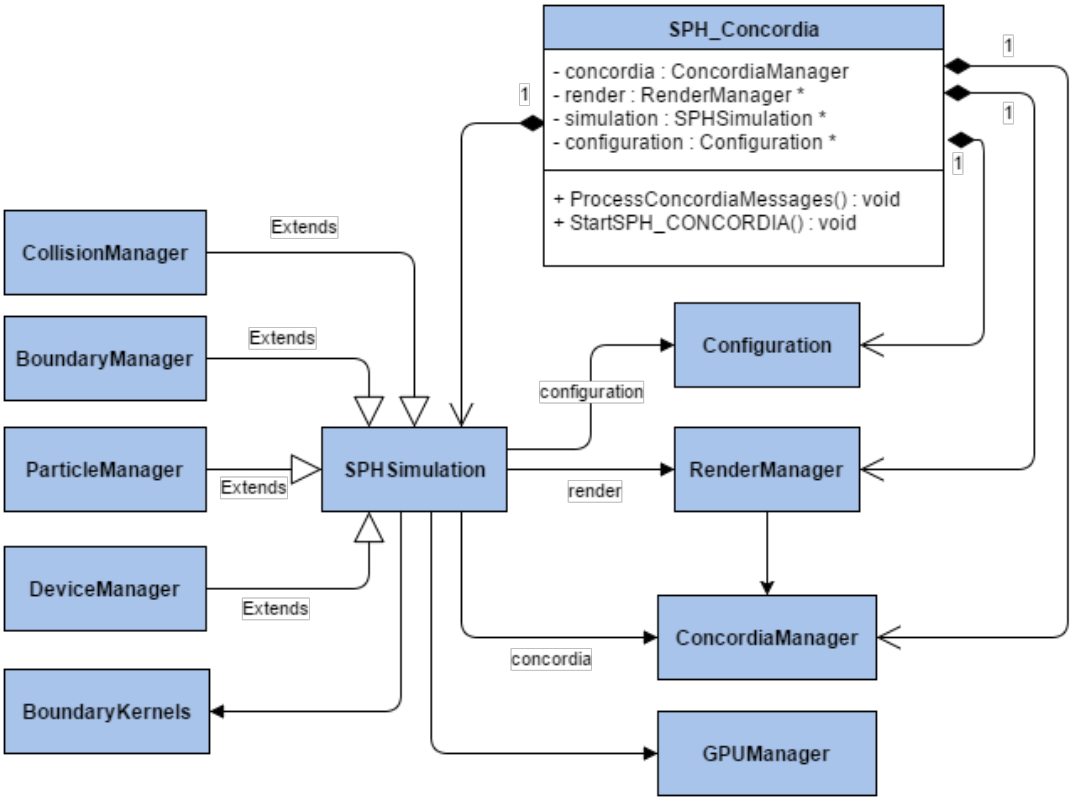

Figure C.5: Diagrama de Clase: SPH_Concordia

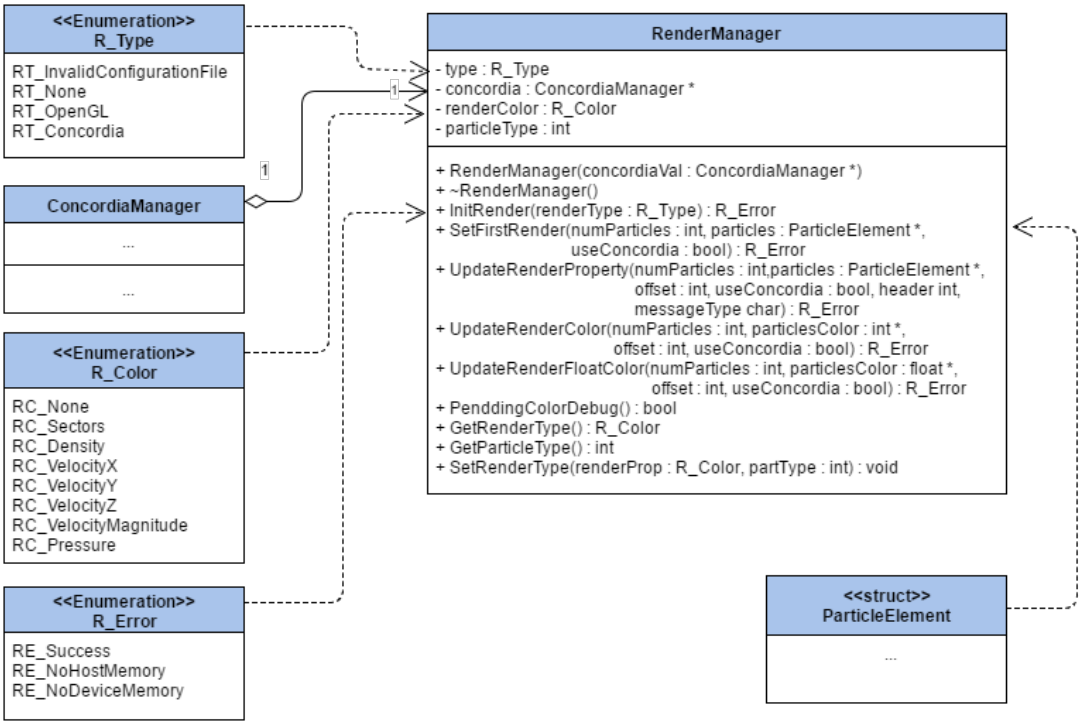

Figure C.6: Diagrama de Clase: RenderManager

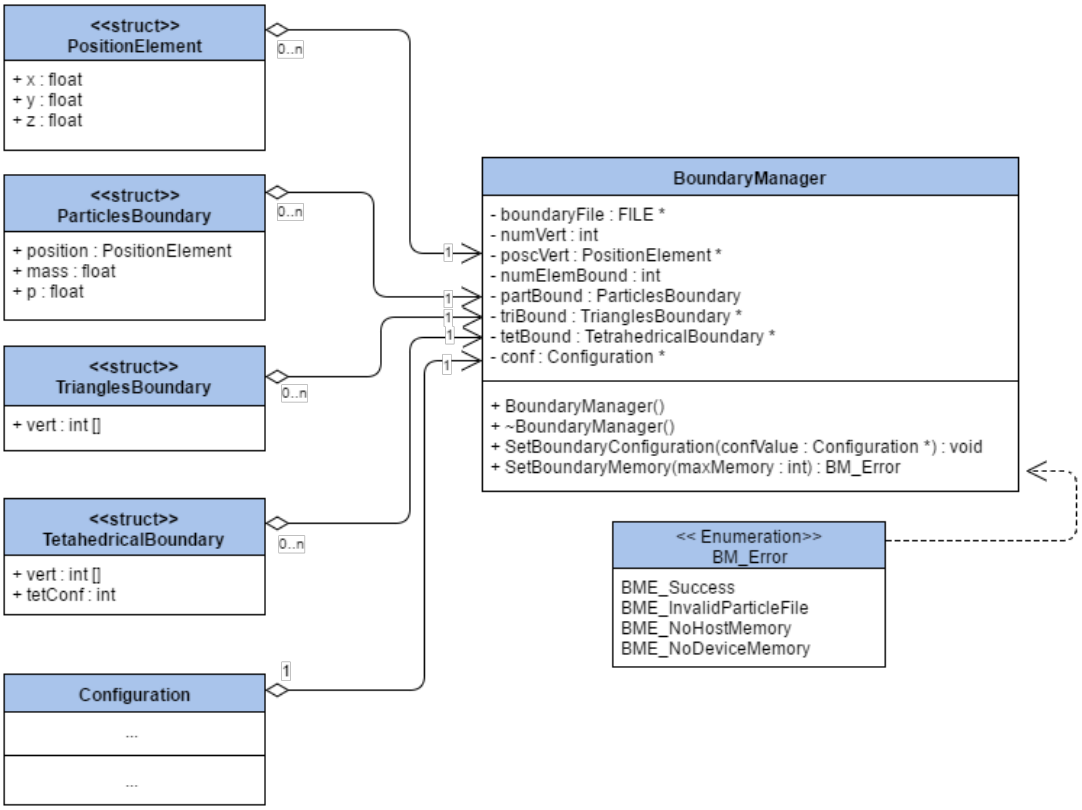

Figure C.7: Diagrama de Clase: BoundaryManager

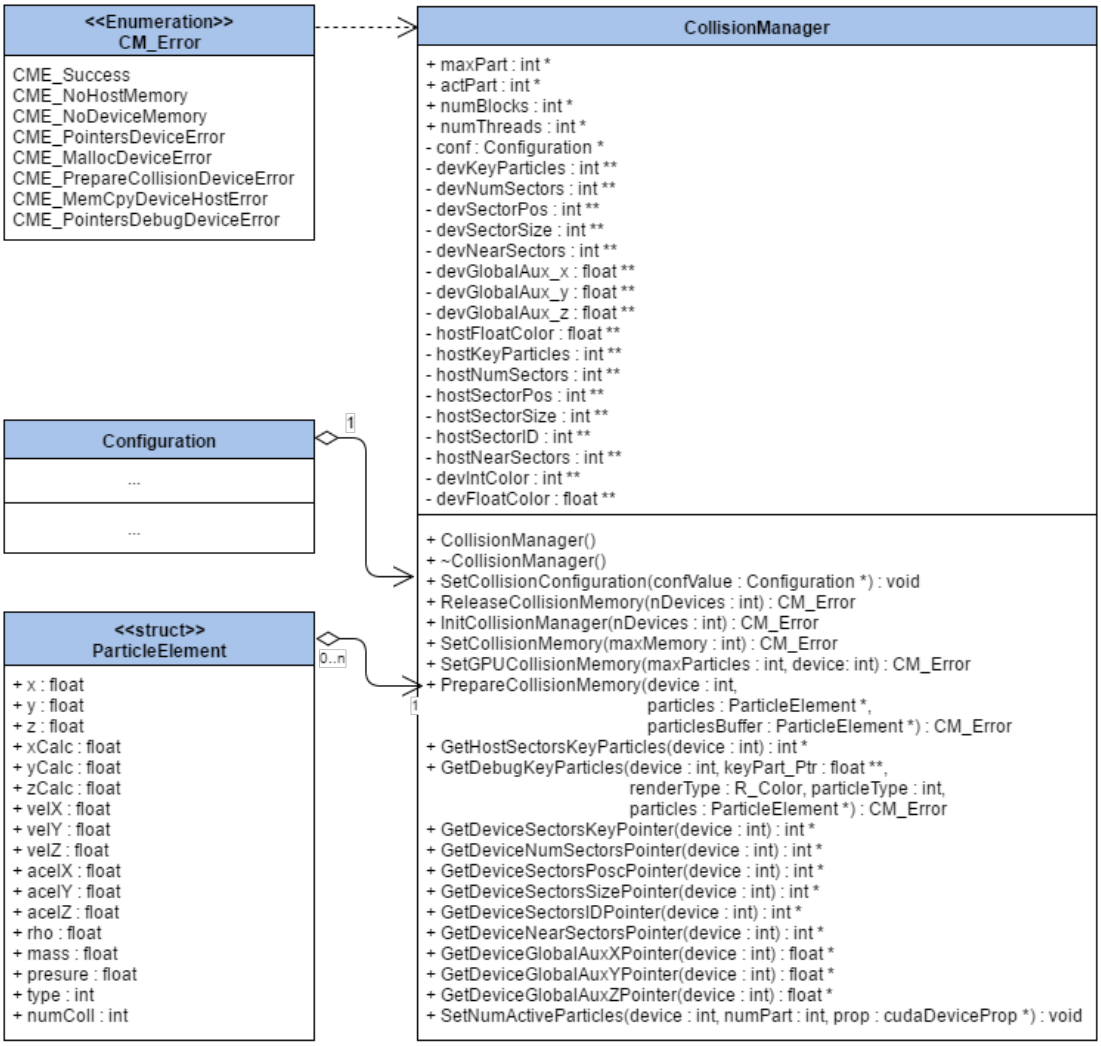

Figure C.8: Diagrama de Clase: CollisionManager

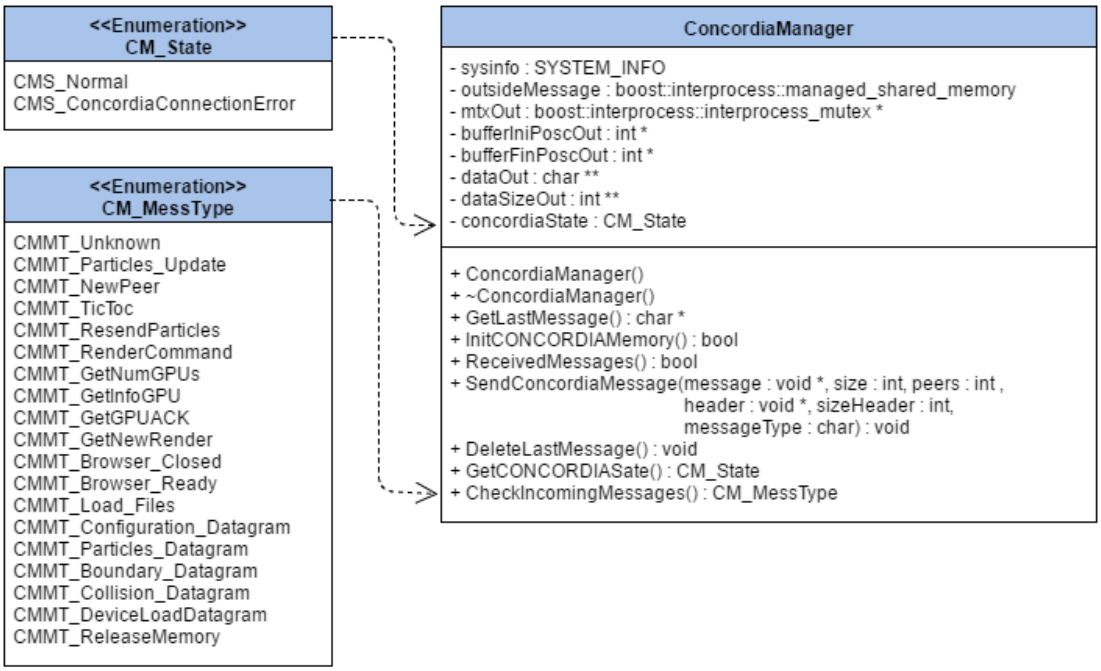

Figure C.9: Diagrama de Clase: ConcordiaManager

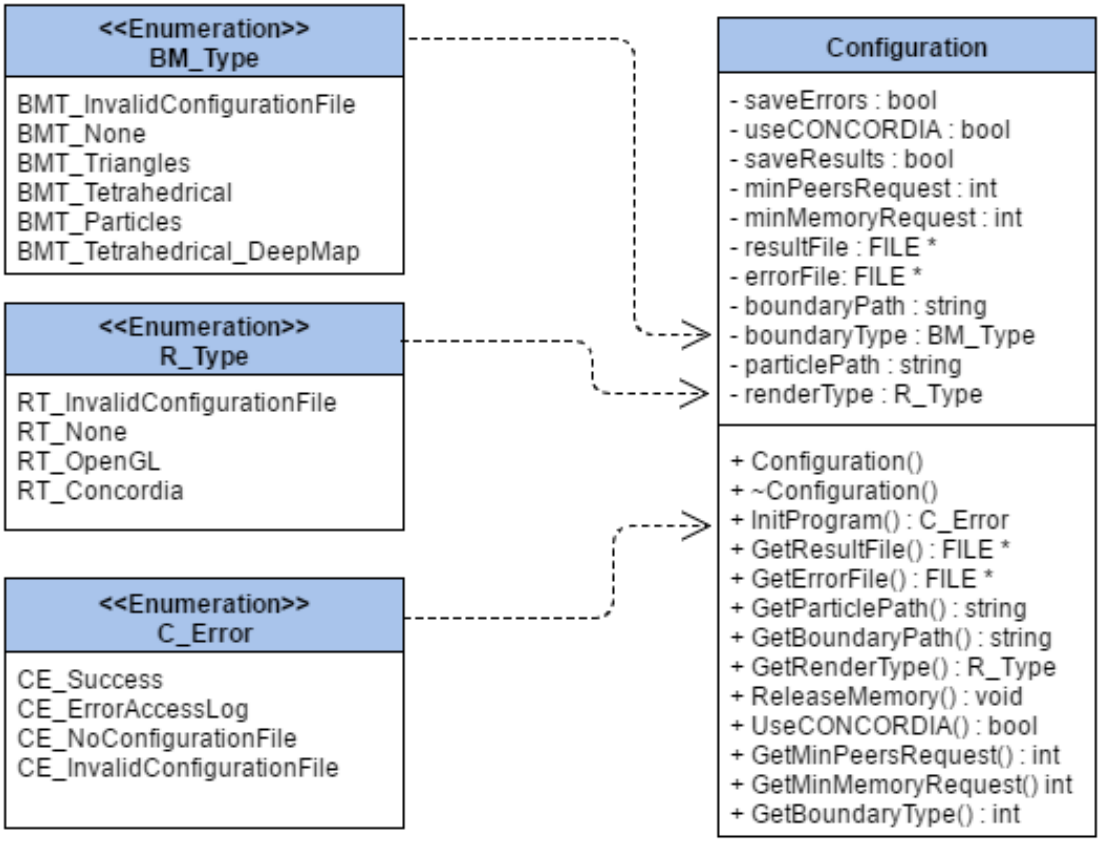

Figure C.10: Diagrama de Clase: Configuration

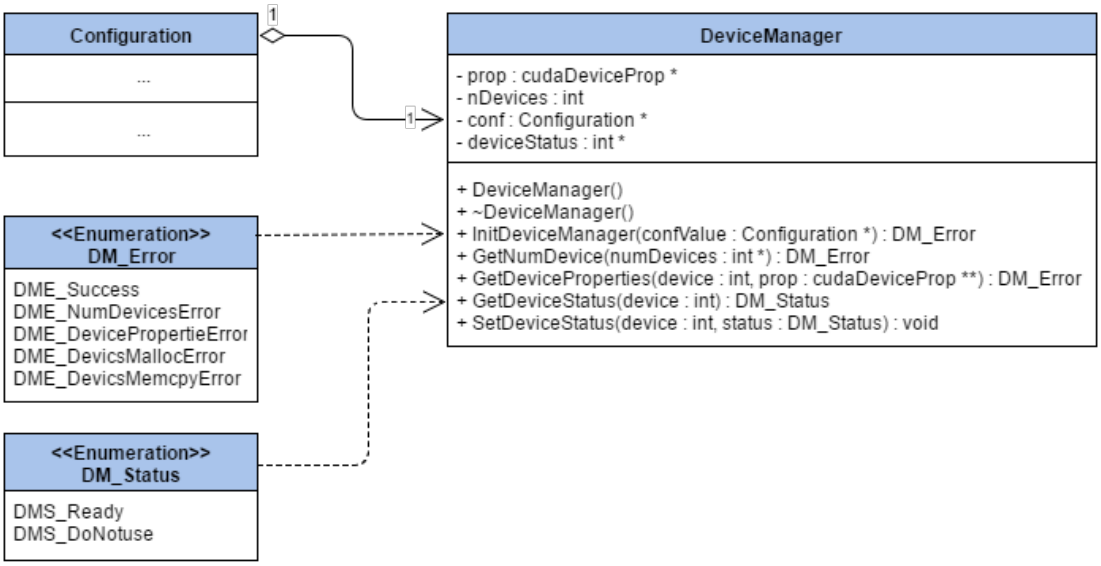

Figure C.11: Diagrama de Clase: DeviceManager

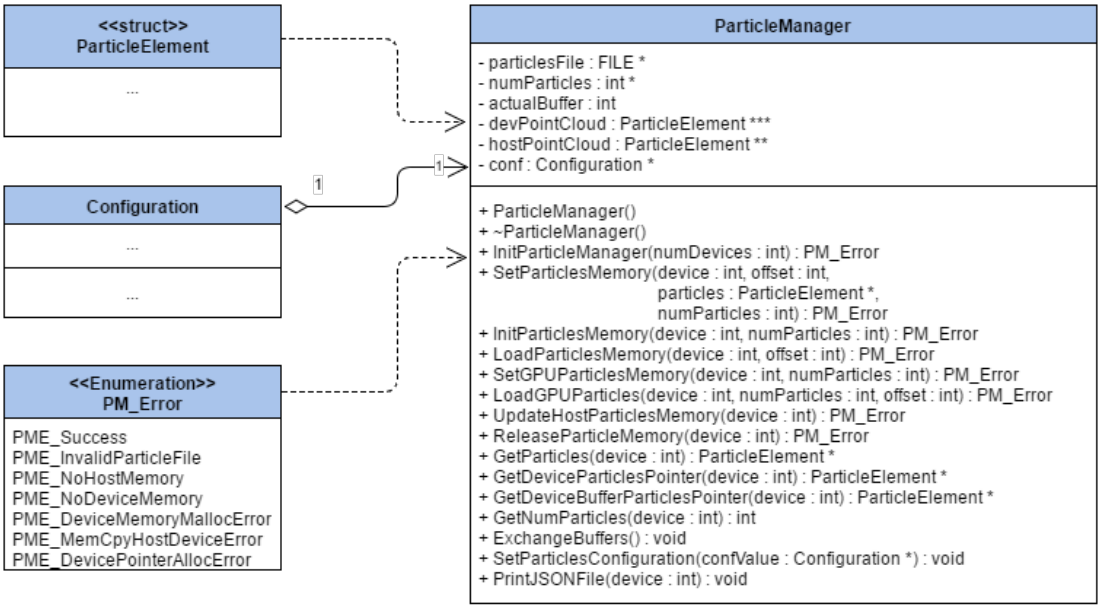

Figure C.12: Diagrama de Clase: ParticleManager

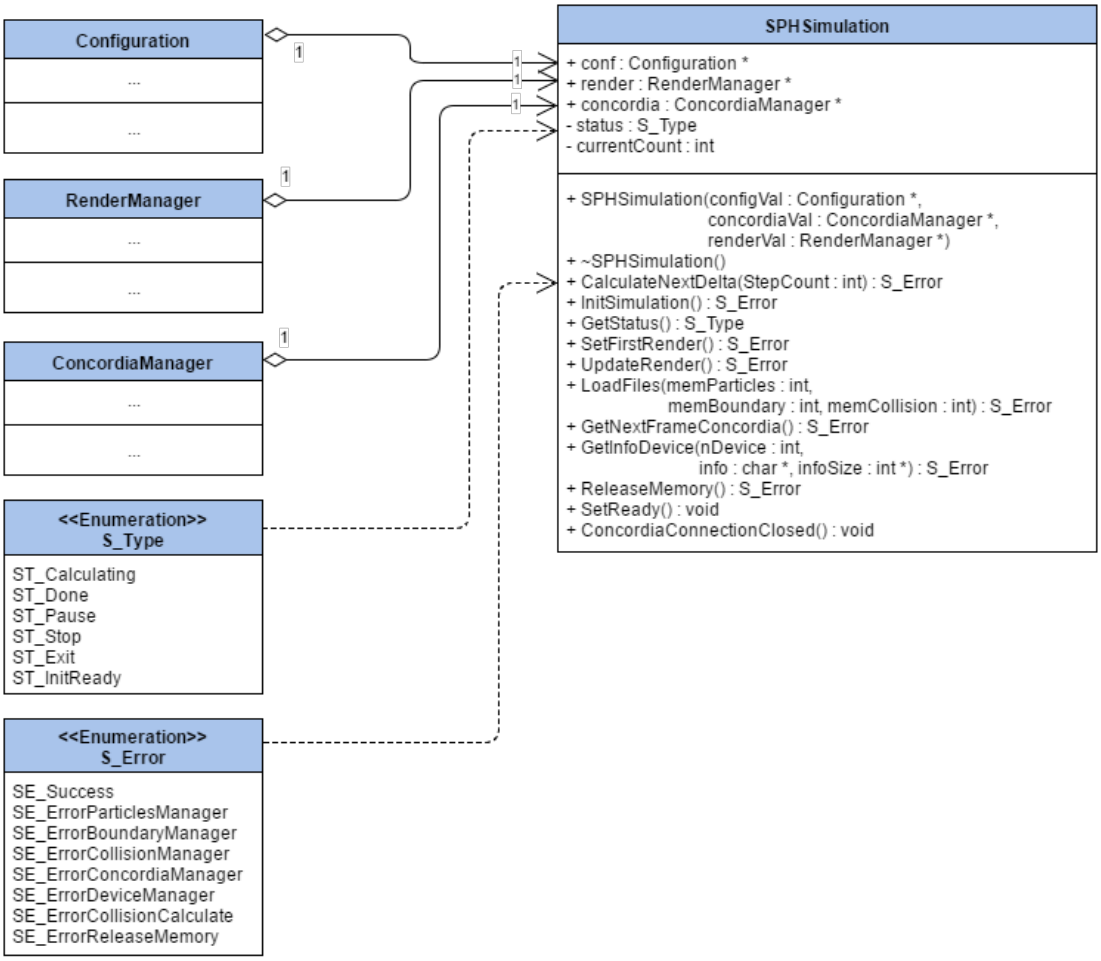

Figure C.13: Diagrama de Clase: SPHSimulation

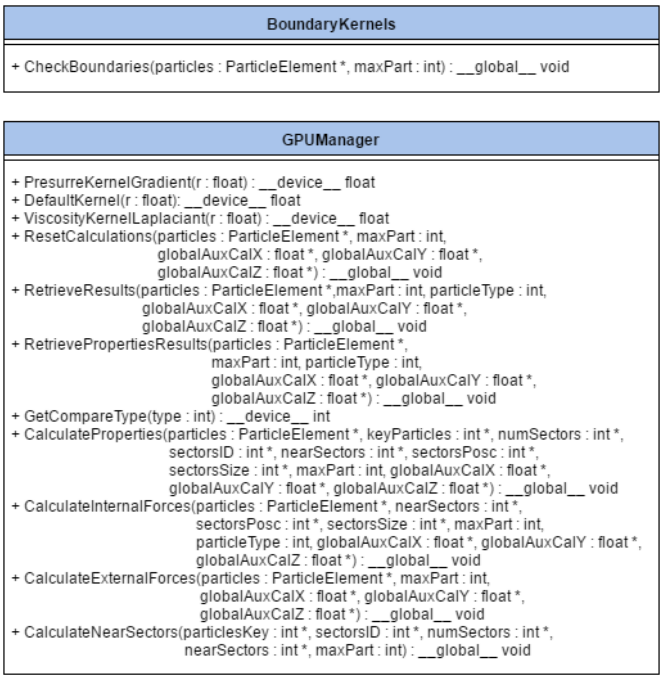

Figure C.14: Diagrama de Clase: GPUManager

C.2 CONCORDIA

Al igual que con el simulador se presenta la máquina de estado y diagrama de clases mostrando las clases necesarias para poder realizar los estados indicados.

C.2.1 Máquina de Estados

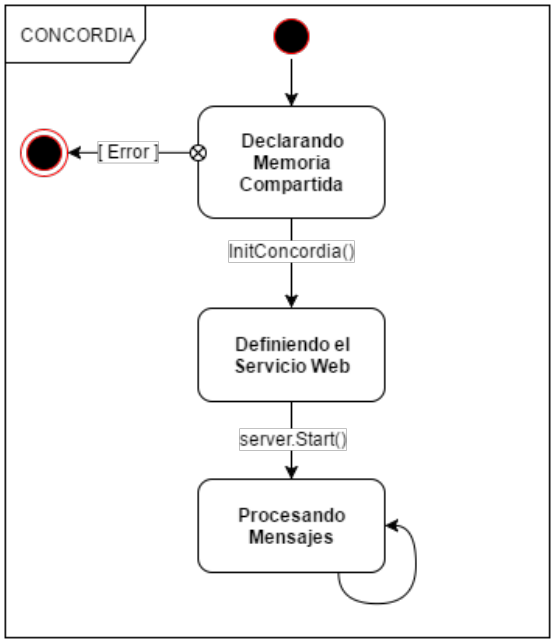

Figure C.15: Máquina de Estados : CONCORDIA

C.2.2 Diagramas de Clases

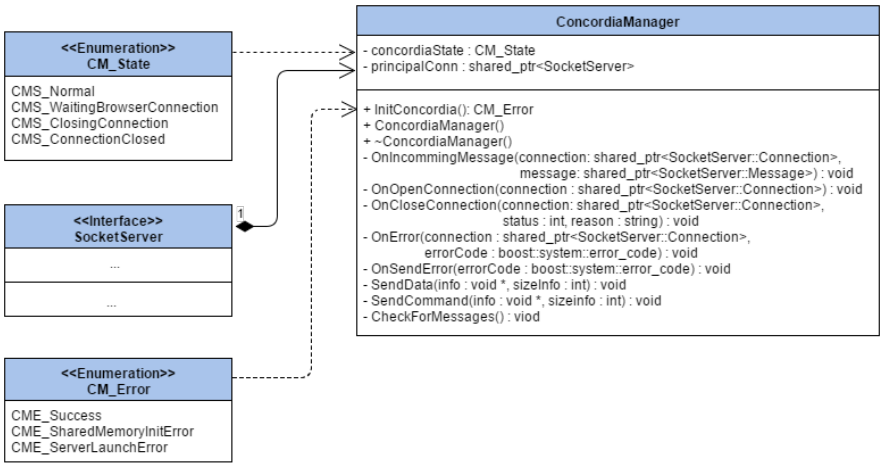

Figure C.16: Diagrama de Clase: ConcordiaManager

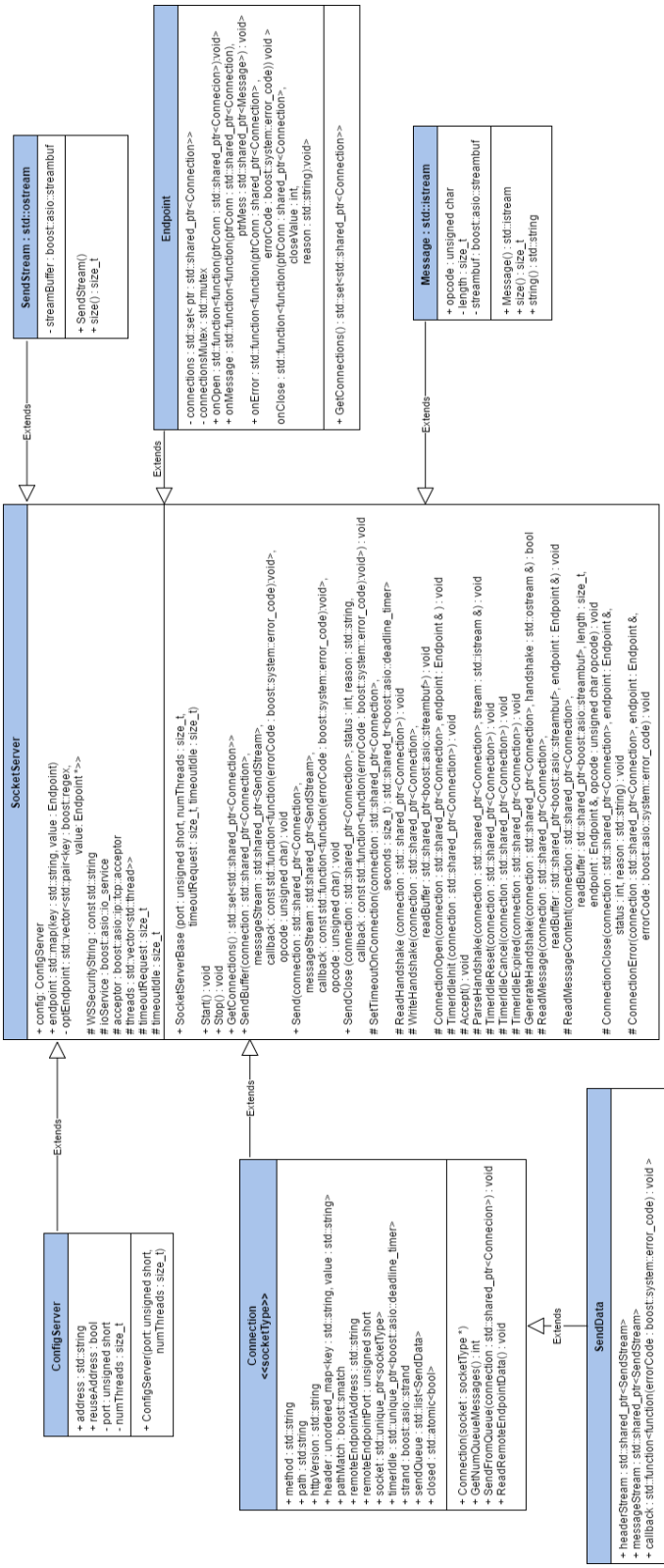

Figure C.17: Diagrama de Clase: SocketServer

Bibliografía

[1] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. *Particle-based Viscoelastic Fluid Simulation.* In Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2005. 1

[2] Nick Foster and Ronald Fedkiw. *Practical Animation of Liquids.* In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, pages 23–30, New York, NY, USA, 2001. ACM. 1, 21

[3] Nick Foster and Dimitris Metaxas. *Modeling the Motion of a Hot, Turbulent Gas.* In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '97, pages 181–188, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 1

[4] Tolga G. Goktekin, Adam W. Bargteil, and James F. O'Brien. *A Method for Animating Viscoelastic Fluids.* ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2004), 23(3):463–468, 2004. 1

[5] Jos Stam. *Stable Fluids.* In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 1

[6] Andrea Colagrossi and Maurizio Landrini. *Numerical simulation of interfacial flows by smoothed particle hydrodynamics.* Journal of Computational Physics, 191(2):448 – 475, 2003. 1, 34

[7] T. Anita Layton and Michiel van de Panne. *A numerically efficient and stable algorithm for animating water waves.* The Visual Computer, 18(1):41–53, 2002. 1

[8] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. *Animating Lava Flows.* In Proceedings of the 1999 Conference on Graphics Interface '99, pages 203–210, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. 1, 21

[9] J. J. Monaghan. *Smoothed Particle Hydrodynamics.* Annual Review of Astronomy and Astrophysics, 30(1):543–574, 1992. 1, 22, 34

[10] Giuseppe Bilotta Alexis Hérault and Robert Anthony Dalrymple. *SPH on GPU with CUDA*. Journal of Hydraulic Research, 48:74–79, Jun 2010. 2

[11] Alberto Cremonesi Luigi Tavazzi and Fausto Castriota. *Carotid Stenting Complications*. e-journal of the ESC Council for Cardiology Practice, 8(17), Jun 2010. 3

[12] *Evaluation of Clinical Data -A Guide For Manufacturers and Notified Bodies*. Technical report, Consumer goods Cosmetics and Medical Devices, Dec 2008. 3

[13] U.S. Department of Health and Human Services. *Non-Clinical Engineering Tests and Recommended Labeling for Intravascular Stents and Associated Delivery Systems*. Technical report, U.S. Department of Health and Human Services, Food and Drug Administration, Center for Devices and Radiological Health, Apr 2010. 18. 3

[14] Paul Morris, Andrew Narracott, Hendrik von Tengg-Kobligk, Daniel Alejandro Silva Soto, Sarah Hsiao, Angela Lungu, Paul Evans, Neil Bressloff, Patricia Lawford, D. Rodney Hose, and Julian P Gunn. *Computational fluid dynamics modelling in cardiovascular medicine*. Cardiovascular Journal Open Heart, 102(1):18 – 28, Oct 2015. 3, 21

[15] Xinyi Leng, Fabien Scalzo, Albert K Fong, Mark Johnson, Hing Lung Ip, Yannie Soo, Thomas Leung, Liping Liu, Edward Feldmann, Ka Sing Wong, and David S Liebeskind. *Computational fluid dynamics of computed tomography angiography to detect the hemodynamic impact of intracranial atherosclerotic stenosis*. Neurovascular Imaging, 1(1):1–7, 2015. 3

[16] Xinyi Leng, Fabien Scalzo, Hing Lung Ip, Mark Johnson, Albert K. Fong, Florence S. Y. Fan, Xiangyan Chen, Yannie O. Y. Sooand Zhongrong Miao, Liping Liu, Edward Feldmann, Thomas W. H. Leung, David S. Liebeskind, and Ka Sing Wong. *Computational Fluid Dynamics Modeling of Symptomatic Intracranial Atherosclerosis May Predict Risk of Stroke Recurrence*. PLoS One, 9(5), May 2014. 3

[17] Selim G. Akl. *Bitonic Sort*, pages 139–146. Springer US, Boston, MA, 2011. 4

[18] Goernig Matthias, Schroeder Rico, Kleindienst Robby, Figulla Hans Reiner, Voss Andreas, and Leder Uwe. *Blood pressure variability analysis enhances risk stratification in chronic heart failure*. Herbert Open Access Journals, Cardiovascular System, 2(5), 2014. 7

[19] Saeid Golbidi and Ismail Laher. *Exercise and the Cardiovascular System*. Cardiology Research and Practice, 2012, 2012. 7

[20] Wintrobe Maxwell Myer. *Wintrobe's clinical hematology*. 2009.

[21] Francis H. Harlow and J. Eddie Welch. *Numerical Calculation of Time Dependent Viscous Incompressible Flow of Fluid with Free Surface.* Phys. Fluid, 8:2182–2189, 1965. 20

[22] T. Belytschko, Y. Y. Lu, and L. Gu. *Element-free Galerkin methods.* International Journal for Numerical Methods in Engineering, 37(2):229–256, 1994. 20

[23] Jens Struckmeier Dietmar Hietel, Konrad Steiner. *A Finite-Volume Particle Method For Compressible Flows.* Math. Models Methods Appl. Sci, 10(2), 2000. 20

[24] R. A. Gingold and J. J. Monaghan. *Smoothed Particle Hydrodynamics: theory and application to non-spherical stars.* Monthly Notices of the Royal Astronomical Society, 181:375–389, 1977. 21

[25] S. Koshizuka, H. Tamako, and Y. Oka. *A particle method for incompressible viscous flow with fluid fragmentation.* 1995. 21

[26] Fabricio Macia Lang, Antonio Souto Iglesias, Matteo Antuono, and A. Colagrossi. *Benefits of using a Wendland Kernel for free-surface flows.* In Proceedings of 6th ERCOFTAC SPHERIC workshop on SPH applications, pages 30–37, 2011. 23

[27] M.J.H. Anthonissen S.P. Korzilius, W.H.A. Schilders. *An improved corrective smoothed particle method approximation for second-order derivatives.* In 8th International Smoothed Particle Hydrodynamics European Research Interest Community Workshop, pages 38–43, 2013. 23

[28] Walter Dehnen and Hossam Aly. *Improving convergence in smoothed particle hydrodynamics simulations without pairing instability.* Mon. Not. R. Astron. Soc, 2012. 27

[29] Matthias Müller, David Charypar, and Markus Gross. *Particle-based fluid simulation for interactive applications.* In Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 154–159. Eurographics Association, 2003. 30

[30] Mathieu Desbrun and Marie-Paule Gascuel. *Smoothed Particles: A new paradigm for animating highly deformable bodies*, pages 61–76. Springer Vienna, Vienna, 1996. 31

[31] Walter Dehnen and Hossam Aly. *SPH modelling of waves.* ASCE Coastal Dynamics, pages 779–787, 2001. 34

[32] Steven S. Skiena. *The Algorithm Design Manual*, chapter Data Structures. 2013. 37

[33] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing.* Technical Report Ottawa, Ontario, Canada, 1966. 41

[34] David Goldberg. *What Every Computer Scientist Should Know About Floating-point Arithmetic*. ACM Comput. Surv., 23(1):5–48, March 1991. 42, 46

[35] Donald Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, chapter Chapter 4.1. 1997. 42

[36] W. Daniel Hillis and Guy L. Steele. *Data Parallel Algorithms*. Commun. ACM, 29:1170–1183, 1986. 50

[37] Mozilla Google and Opera Developers. *WebRTC Project*, June 2016. 57

[38] Alexey Melnikov Ian Fette. *The WebSocket Protocol*, December 2011. 58

[39] Michelle Bu and Eriz Zhang. *The PeerJS Library*, June 2014. 60

[40] Ricardo Cabello. *Three.js*, April 2010. 61

[41] Gene M. Amdahl. *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*. In Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. 64

[42] JAKOB NIELSEN. *Nielsen's Law of Internet Bandwidth*, April 1998. 72

[43] Intel. *Moore's Law*, June 2016. 72

[44] Oculus. *Oculus VR Devices*, June 2016. 72

[45] Google. *Google Cardboard*, June 2016. 72

[46] NVIDIA. *http://www.nvidia.com/page/products.html*, June 2016. 81

[47] Hwu Stratton, Stone. *CUDA programming guide 2.3*. Technical Report Santa Clara, CA : N, 2008. 89

[48] Object Management Group (OMG). *Documents Associated With Unified Modeling Language (UML) Version 2.5*. OMG Document Number formal/2015-03-01 (`http://www.omg.org/spec/UML/2.5`), 2015. 95