ESCUELA POLITÉCNICA SUPERIOR DE JAÉN

Grado en Ing. Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Optimización de algoritmos geométricos con programación hardware

Alumno: Mario Salazar de Torres

Tutor: Juan Ruiz de Miras
Dpto: Informática

Junio, 2016

Juan Ruiz de Miras como tutor del Trabajo Fin de Grado *Optimización de algoritmos geométricos con programación hardware*, autoriza su presentación para defensa y evaluación a **Mario Salazar de Torres** en la Escuela Politécnica Superior de la Universidad de Jaén.

Jaén, x de Junio de 2016

El alumno: El tutor:

Mario Salazar de Torres Juan Ruiz de Miras

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 PREFACIO

1.1 MOTIVACIÓN

En geometría computacional se suele tratar con problemas con grandes cantidades de datos, los cuales suponen un alto tiempo de procesamiento para algoritmos con complejidad igual o superior a O(n).

Debido a la creciente tendencia por parte de las aplicaciones de reducir el tiempo de ejecución de sus procesamientos, más aun cuando hablamos de aplicaciones gráficas, es una necesidad imperante la búsqueda de soluciones alternativas a la ejecución secuencial no-simultanea de los últimos años.

En nuestro caso el problema que se nos plantea es la resolución de uno de los problemas más básicos de la geometría computacional, la comprobación de la inclusión de un punto en un politopo. Este problema es igualmente básico y complejo, pues uno de los problemas que presenta su resolución es la gran cantidad de casuísticas que se pueden presentar.

Dicho problema es la base de muchas otras soluciones como por ejemplo, la voxelización de una representación B-Rep (1) de un modelo, la inclusión de un punto en un sólido de forma libre, la ejecución de un *ray casting* (2) o incluso tiene su aplicación en otros campos como la psicología, en el análisis n-dimensional de tendencias.

Tras el auge de los estándares SIMD, los proyectos de investigación en procesadores de alto rendimiento y la aparición del paradigma GPGPU, así como últimamente la producción de procesadores cuánticos adiabáticos, nuestro enfoque debido a la potencia y facilidad de adquisición del hardware necesario es el de GPGPU.

Durante los últimos años hemos podido ver como una gran cantidad de investigadores de la computación han ido abandonando los tradicionales paradigmas para plasmar sus algoritmos mediante el paradigma GPGPU. Hemos visto la aplicación de este paradigma en campos tan amplios como la medicina, la física, las matemáticas, psicología, química, aerodinámica, investigación militar, inteligencia artificial, etc.

En definitiva el motivo de la realización de este proyecto es demostrar cómo es posible aplicar este paradigma a algoritmos geométricos de forma eficiente con resultados bastante buenos. De hecho a casi cualquier algoritmo se le puede aplicar el paradigma GPGPU, aunque la ganancia en tiempo de ejecución dependerá de la implementación y el grado en que el algoritmo permita ser paralelizado.

1.2 PROPÓSITO

El propósito del proyecto es el de analizar el estado del arte de los demás algoritmos existentes para la resolución del problema de la inclusión de punto en politopo, desarrollar un nuevo algoritmo que mejore a los anteriores en la medida de lo posible.

1.3 OBJETIVOS

- Revisión del estado del arte de los demás algoritmos de inclusión de un punto en un politopo.
- Desarrollo e implementación en GPGPU (CUDA) de un nuevo algoritmo basado en el descrito en el artículo de inclusión de punto en sólido de forma libre (3).
- Implementación del anterior algoritmo mediante el paradigma de programación monohilo y multi-hilo sobre CPU, con el objetivo de realizar las oportunas comparaciones.
- Ejecución y recogida de estadísticas de las distintas versiones del nuevo algoritmo desarrollado.
- Implementación de otras versiones GPGPU del algoritmo que solucionen el mismo problema.
- Ejecución y recogida de estadísticas de los algoritmos de comparación.
- Comparación de los resultados obtenidos en las diferentes ejecuciones, llegando a una conclusión acerca del tiempo de ejecución de las distintas implementaciones de los algoritmos con sus respectivos motivos.
- Demostrar la usabilidad de los algoritmos GPGPU mediante una aplicación práctica como es la voxelización de modelos CSG (4) a partir de primitivas B-Rep (1).

1.4 METODOLOGÍA

Dada la poca convencionalidad de la tipología del proyecto la metodología a seguir no puede ser similar a la de un proyecto de ingeniería tradicional. La metodología a seguir en este proyecto será:

- Se desarrollará un estudio teórico y una revisión del estado del arte de los algoritmos geométricos a optimizar.
- Se establecerán las tecnologías de programación GPU/CPU multi-núcleo y las plataformas hardware a utilizar en función del algoritmo a optimizar.
- Se implementarán los algoritmos en las plataformas hardware-software seleccionadas.
- Se realizará un análisis y una discusión de los resultados obtenidos.
- Se detallará en la memoria del trabajo fin de grado todo el proceso seguido y los resultados obtenidos, incluyendo las conclusiones del trabajo realizado y las referencias bibliográficas utilizadas.

1.5 RESULTADOS

- Aplicaciones correspondientes a las distintas implementaciones del algoritmo de inclusión de punto en politopo.
- Modelos 3D utilizados durante las ejecuciones de los algoritmos.
- Aplicaciones de utilidad para las ejecuciones de los algoritmos.
- Código fuente de la solución contenedora de todas las aplicaciones.
- Manual de usuario de las aplicaciones.
- Memoria.

1.6 PLATAFORMA DE DESARROLLO

En esta sección se pretende dar una visión del marco de trabajo utilizado para la implementación de los algoritmos mediante el paradigma GPGPU

1.6.1 CUDA Toolkit

Para la implementación de los algoritmos en este caso el Framework escogido es el de CUDA Toolkit

CUDA Toolkit es un conjunto de herramientas y utilidades las cuales ayuda al usuario a crear soluciones GPGPU, así como depurar y optimizar dichas soluciones. Este Framework, contiene los siguientes componentes en su última versión:

- **Herramientas:**
  - Contiene un compilador para el lenguaje de que recibe el mismo nombre que el toolkit, estando este basado en C/C++. Dicho compilador se llama nvcc.
  - Entornos de desarrollo como Nsight (Linux) y Nsight VSE (Windows), los cuales ayudan al testeo y optimización.
  - Depuradores como cuda-memcheck, el cual ayuda a localizar los accesos incorrectos, o cuda-gdb y Nsight VSE que te ayudan a trazar paso a paso las aplicaciones.
  - Herramientas de análisis de rendimiento del software producido por el toolkit. Dichas herramientas son nvprof, nvvp y Nsight VSE, y a grandes rasgos te permiten obtener estadísticas de ejecución referentes al tiempo empleado en la lectura/escritura memoria de los distintos tipos de memoria y la ejecución de las instrucciones de la aplicación. Estas herramientas son fundamentales para el proyecto pues ayudan a localizar los puntos a optimizar de los algoritmos, así como los cuellos de botella.
  - Otras utilidades como cudaobjdump y nvdisasm que ayudan al desarrollador en el proceso de depuración.

- **Librerías:**
  - *CUBLAS* es una librería de rutinas de algebra lineal aceleradas mediante CUDA.
  - *CUDA Occupancy* es una librería para el cálculo de la ocupación de un núcleo.
  - *CUDA Runtime API* es una librería de funciones utilitarias.
  - *CUFFT* es una librería para realizar la transformada rápida de Fourier mediante CUDA.
  - *CUPTI* es una interfaz para el análisis de rendimiento del código generador por CUDA.
  - *CURAND* es una librería de generación de números aleatorios mediante algoritmos de alta calidad, estando estos optimizados para su ejecución en CUDA.
  - *CUSOLVER* es una librería de resolución a través de sistemas densos y dispersos lineales, además de para el cálculo de los auto valores de matrices densas y dispersas.
  - *CUSPARSE* es una librería para operar con matrices dispersas implementada para CUDA.
  - *NPP* es una librería para el cálculo de operaciones simples de procesamiento de imágenes y señales, así como operaciones aritméticas, estadísticas y lógicas.
  - *NVBLAS* es una librería que distribuye el procesamiento de cálculos algebraicos lineales entre las GPU y CPU del sistema.
  - NVCUVID es una librería de aceleración para la decodificación de video mediante CUDA. Añade soporte para la decodificación de H.264/AVCHD y MPEG-2/VC-1 en su versión actual.
  - NVRTC fue incorporada en las últimas versiones y es una librería que permite la compilación de código escrito en CUDA en tiempo de ejecución. El objetivo de esta librería es permitir al desarrollador crear un núcleo lo más optimizado en función del hardware objetivo.
  - THRUST es una librería que proporciona soporte para la paralelización de bloques de código. Es comúnmente usado en el desenrollado de bucles de aplicaciones CUDA.
- **Ejemplos.** CUDA Cuenta con un gran repositorio de ejemplos los cuales ayudan al desarrollador a la comprensión de las distintas librerías de CUDA Toolkit.
-

Es importante destacar que estas no son todas las librerías existentes que aceleren su código mediante el uso de CUDA. Muchas librerías oficialmente reconocidas por CUDA como CUDNN (Deep Neural Network) o NVENC (transcodificación de video) han aparecido en los últimos años, aun así no se han incluido, al menos de momento, en CUDA Toolkit.

1.6.2 CUDA Hardware

Para ejecutar las aplicaciones producidas mediante CUDA es obviamente necesario un procesador gráfico, pues que es un lenguaje para la generación de código basado en el paradigma GPGPU.

CUDA no es ni mucho menos un lenguaje para la programación de hardware libre, sino que está únicamente diseñado para los procesadores gráficos de NVIDIA.

En la última década los procesadores gráficos de NVIDIA han ido evolucionando, y en consecuencia las posibilidades que CUDA ofrecía también.

Los avances principales de CUDA vinieron marcados por la aparición de las nuevas arquitecturas de procesadores gráficos de NVIDIA, es por esto que es importante hacer un repaso general de las arquitecturas desde que el primer procesador gráfico con soporte para CUDA apareció.

1.6.2.1 Micro-arquitectura Tesla

Internamente la ejecución de los shaders y los programas realizados con CUDA son bastante similares, aun así esta arquitectura fue la primera en presentar un marco para el desarrollador que desacoplase el hardware de la programación del pipeline para poder procesar cualquier tipo de información de una manera más simple, lo cual supuso un gran impulso para el paradigma GPGPU.

En la Figura 1 se presentan de los componentes básicos del hardware de la arquitectura Tesla involucrado en la ejecución del código escrito en CUDA.

Como se puede observar los procesadores gráficos en la arquitectura se dividen en los multi-procesadores de flujo (SM), que no son más que un conjunto de procesadores de flujo (SP).

Cada uno de estos multi-procesadores consta de unidades multi-hilo de captación y emisión, así como de una memoria compartida y una memoria constante. Además tienen varias unidades de ejecución para instrucciones espciales.

Los multi-procesadores están conectados a un nivel superior con la cache de texturas y a traves de una red de interconexión con la memoria global.

Es importante destacar que desde sus principios CUDA define una unidad mínima de ejecución llamada warp. Estos, son grupos de procesadores de flujo que ejecutan la misma instrucción en el mismo ciclo.

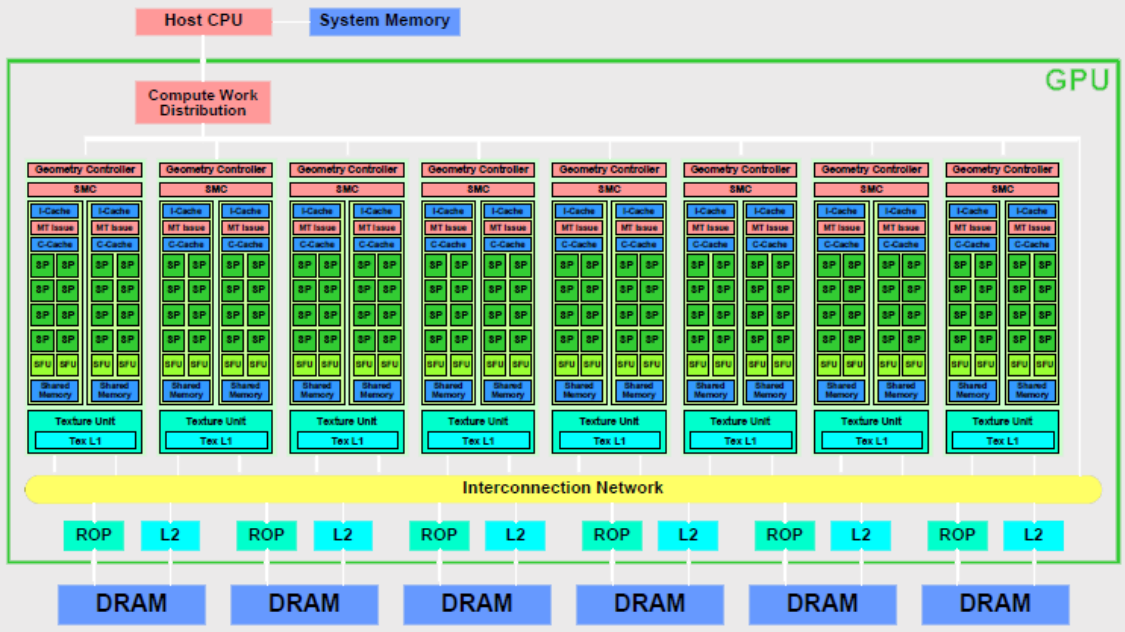

Figura 1. Micro-arquitectura Tesla. GPU G80 de NVIDIA

Uno de los factores claves en CUDA es el manejo de la memoria, es crucial conocer que tipos de memoria hay y que partes del hardware pueden acceder a cada tipo de memoria, para así poder desarrollar aplicaciones en CUDA lo más eficientes posible. Los tipos son:

- Registros. Cada procesador de flujo tiene un conjunto de registros, los cuales solo pueden ser accedidos desde su mismo procesador de flujo. Este tipo de memoria es el más rápido que existe.
- Memoria local. Esta memoria da acceso a las variables privadas de cada contexto de ejecución de los procesadores de flujo. Está compuesta por registros y memoria global, de tal modo que cuando existen más variables locales simultáneamente que registros, los datos de las variables locales se almacenan en la memoria global.
- Memoria compartida. Es una memoria de baja latencia compartida por cada multi-procesador de flujo. Tiene un alto rendimiento y se suele utilizar cuando la naturaleza del algoritmo requiere realizar acceso a posiciones de memoria vecinas.
- Memoria constante/textura. Es una memoria de solo lectura y tiene una latencia de acceso media.
- Memoria global. Es una memoria con una muy alta latencia de acceso y es común a todos los multi-procesadores de flujo.
- Cache L2. Este tipo de memoria reduce el acceso a las variables almacenadas en memoria global.

1.6.2.2 Micro-arquitectura Fermi

Esta nueva arquitectura vino de mano de la GPU GF100 y el objetivo de la arquitectura era solucionar varios problemas notados en las anteriores micro-arquitecturas. Estos son:

- La falta de soporte para las excepciones.
- La imposibilidad de ejecución de varios núcleos.
- La desorganizada jerarquía de memoria.

En la Figura 2 se muestran los cambios añadidos a los multi-procesadores de flujo en la arquitectura Fermi.

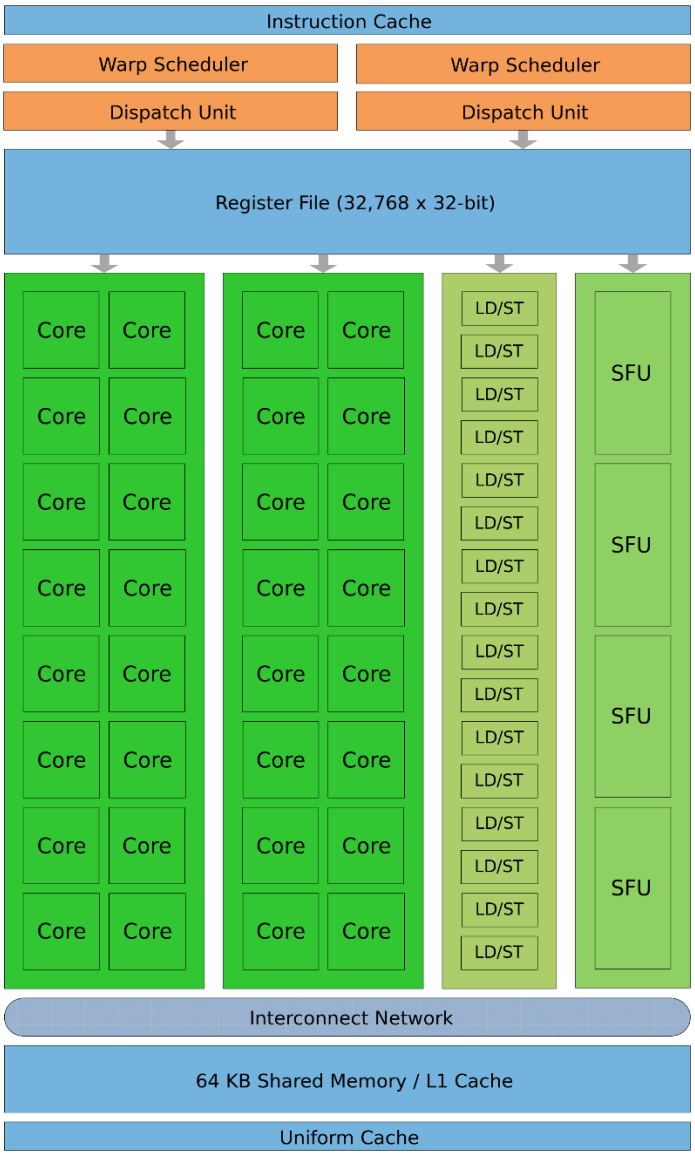

Figura 2. SM de la micro-arquitectura Fermi

En esta nueva micro-arquitectura cada multi-procesador de flujo tiene su propia cache L1.

Además añadió soporte para la memoria gráfica GDDR5, la cual da soporte para hasta un ancho de banda de memoria de 144 GB/s.

En la Figura 3 se puede observar el cambio introducido a la micro-arquitectura en relación la interfaz de memoria y es que cada multi-procesador de flujo tiene su propia unidad de carga y almacenamiento conectada a la red de interconexión de la memoria. Esto supuso una enorme mejora en los tiempos de ejecución.

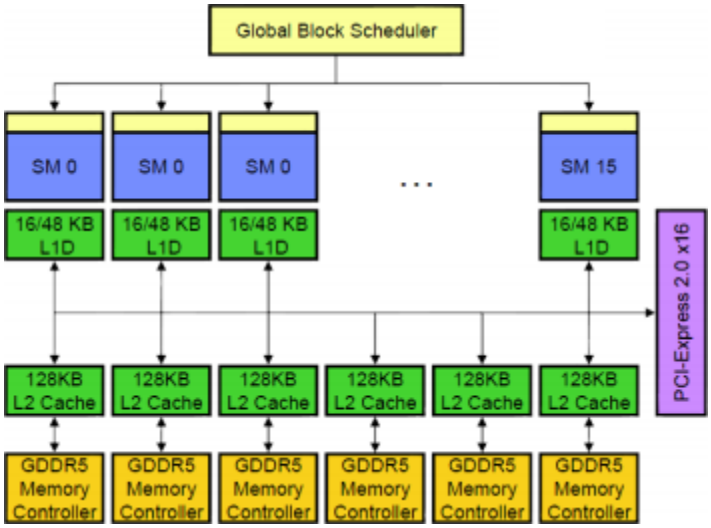

Figura 3. Micro-arquitectura Fermi

En esta nueva micro-arquitectura el multi-procesador de flujo se rediseñó. Tal y como se ve en la Figura 4 se añade un registro de las instrucciones en ejecución de un warp con el objetivo de añadir un mayor desacople que suponga mayor nivel de paralelización.

Cada multi-procesador de flujo dispone de dos puertos, los cuales pueden acceder bien a la FPU o a la ALU, pero no a ambos.

Dichos puertos tienen 16 unidades de ejecución cada uno, por lo que en esta micro-arquitectura cada warp consume 2 ciclos en completarse.

Además Fermi fue la primera micro-arquitectura en incluir detección de errores en memoria.

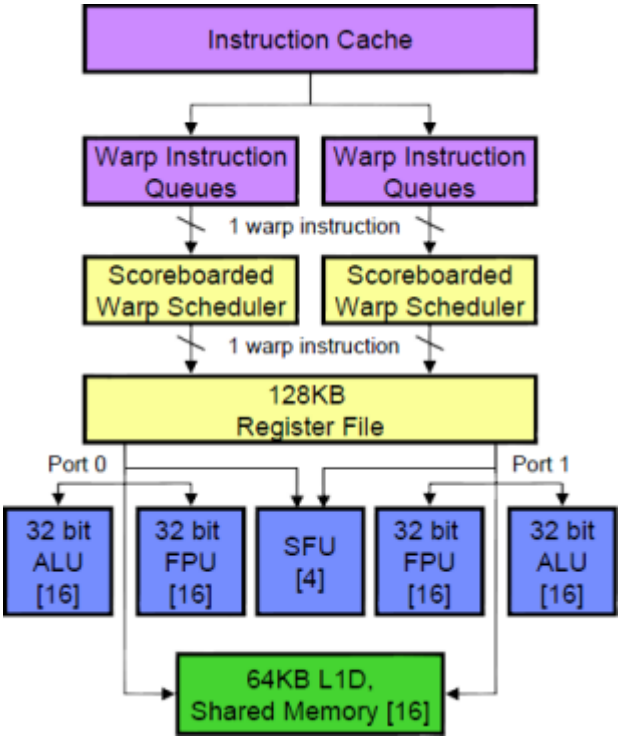

Figura 4. Detalle del SM de la micro-arquitectura de Fermi

1.6.2.3 Micro-arquitectura Kepler

El primer chip de la micro-arquitectura (GK104) no introdujo realmente avances significativos en rendimiento gráfico. El objetivo de este chip fue el de aumentar el rendimiento por vatio al mismo tiempo que se mantiene la misma potencia gráfica, lo que supuso por tanto, una reducción en el consumo del chip.

Como se puede ver en la Figura 5 la micro-arquitectura Kepler organiza sus multi-procesadores de flujo en clústeres de procesamiento gráfico, al mismo modo que lo hacía Fermi.

Uno de los mayores avances de Kepler fue la introducción de los multi-procesadores de flujo de siguiente generación (SMX), los cuales introducen una variedad de cambios sobre el antiguo multi-procesador de flujo.

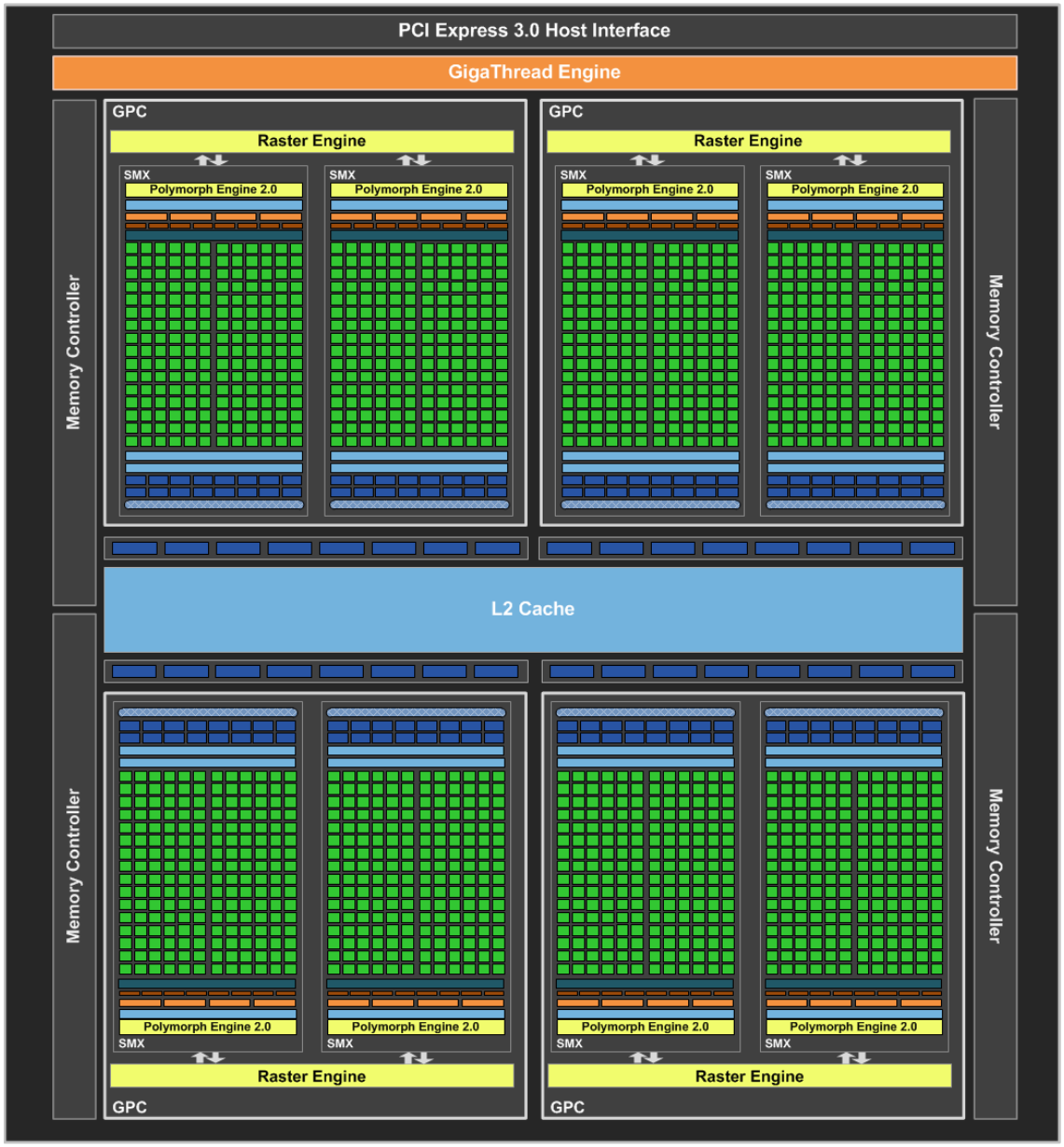

Figura 5. Micro-arquitectura Kepler. GTX 680 (GK104)

Uno de los cambios significativos de Kepler es la supresión del registro de estado de los registros de memoria que impedían que ocurriesen incoherencias matemáticas debido a la concurrencia de los procesadores gráficos, tal y como se puede ver en la Figura 6.

Ya que todas las instrucciones tienen una latencia constante, la necesidad del anterior mencionado registro era innecesaria y podía relegarse la integridad de los resultados de las ejecuciones al compilador, el cual se encargaría de organizar la ejecución de las operaciones.

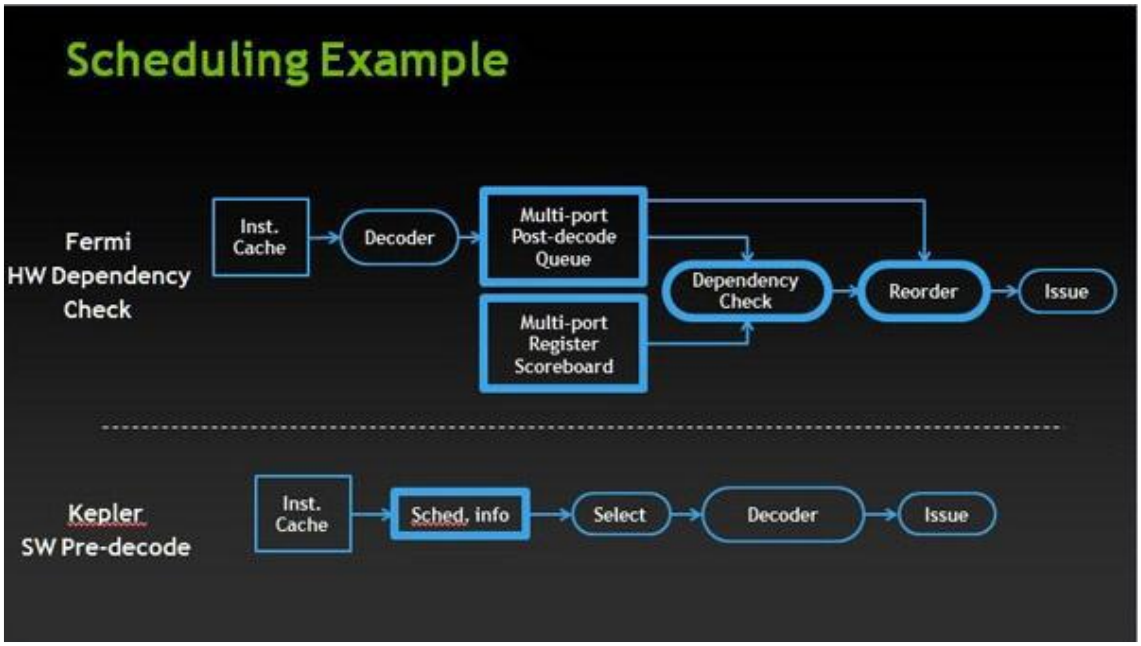

Figura 6. Ejecución de instrucciones en Fermi/Kepler

Otra significativa mejora en Kepler es el cambio de la cache L2, lo cual supuso un incremento en la tasa de aciertos en un 73%.

Además las operaciones atómicas incrementaron significativamente el rendimiento.

GK104 supuso muchas otras mejoras, aunque no las describo aquí, pues no tuvieron un impacto directo en CUDA.

El GK110 marcó un antes y un después en la micro-arquitectura Kepler, pues supuso cambios como:

- Aumento del rendimiento de las operaciones en coma flotante de doble precisión.
- La implementación de nuevas operaciones shuffle.
- Mejora de las operaciones atómicas sobre la memoria global.
- Soporte de las operaciones atómicas de 64-bits.
- Mayor flexibilidad en la elección del tamaño de la memoria compartida y la cache L1.
- La introducción del paralelismo dinámico.
- La inclusión de Hyper-Q.
- Una nueva característica denominada GPUDirect.

El paralelismo dinámico es una de las nuevas características de GK110, la cual permite la ejecución de núcleos directamente desde la GPU.

Muchos de los algoritmos ejecutados por la GPU son completamente paralelos, sin embargo existen algoritmos paralelos con partes potencialmente recursivas.

Un ejemplo clarificador es la de la segmentación de una imagen, tal y como se muestra en la Figura 7. Mediante paralelismo GPGPU convencional es perfectamente realizable, sin embargo la introducción del paralelismo dinámico supone una gran mejora de rendimiento en estos algoritmos.

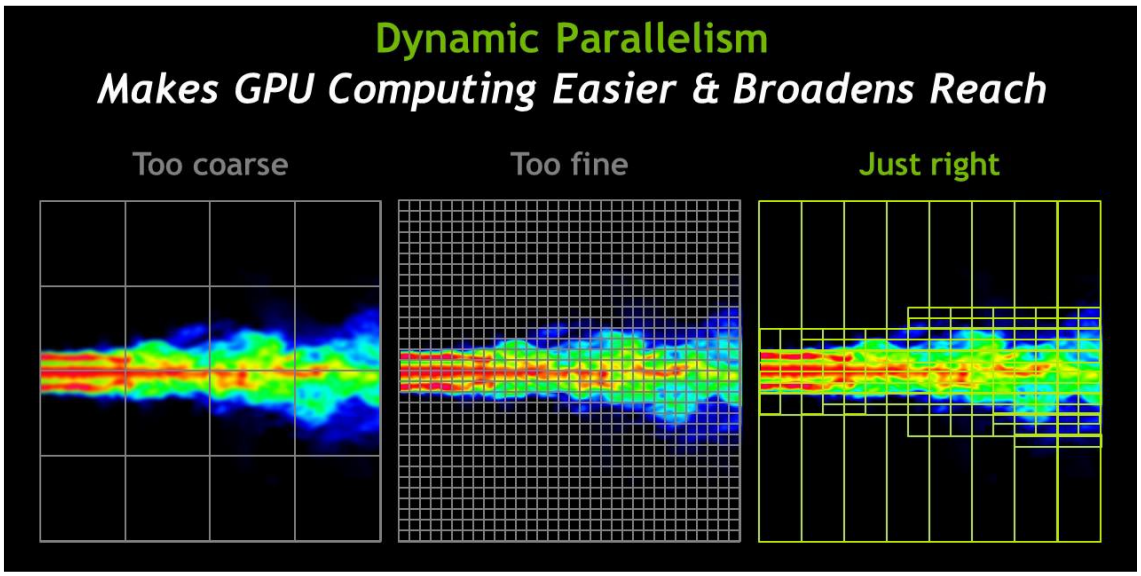

Figura 7. Segmentación de una imagen con paralelismo dinámico

Hyper-Q es otra característica que apareció con el chip GK110, la cual supone un impacto altamente positivo en las aplicaciones que ejecutan una intensa cantidad de trabajo al mismo instante.

Esto es debido a que esta característica incrementando a 32 las conexiones manejadas por hardware entre el anfitrión y la lógica del distribuidor de trabajo de CUDA (CWD). Esto quiere decir que las operaciones de un flujo de ejecución no bloquearán a los demás flujos.

Es decir, cada flujo de ejecución dispone de su propia cola de trabajo tal y como la Figura 8 muestra.

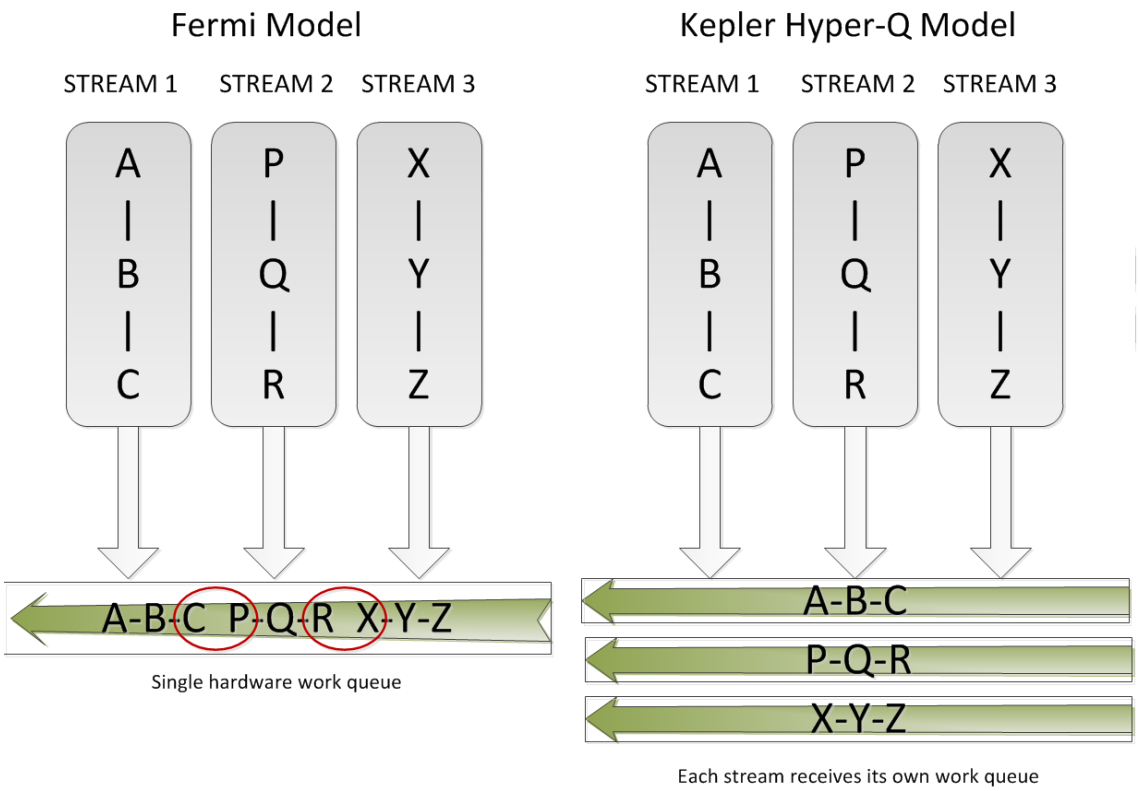

Figura 8. Comparativa en los flujos de ejecución entre Fermi y Kepler

1.6.2.4 Micro-arquitectura Maxwell

GM107 fue el primer chip de la micro-arquitectura y fue fabricado con el objetivo de incrementar el rendimiento por vatio del procesador gráfico.

En este caso manteniendo la misma potencia GM107 apareció con el objetivo de doblar el rendimiento. Para esto fue que modificaron nuevamente los multi-procesadores de flujo.

El multi-procesador de flujo de siguiente generación, SMM, consiguió aún más eficiencia energética y dio como resultado hasta un 35% más de rendimiento por procesador de flujo.

Esto es debido a los cambios introducidos en la arquitectura y algoritmos del organizador del multi-procesador de flujo, de tal modo que se evitan bloqueos innecesarios.

Otro de los cambios introducidos fue en la memoria, pues incrementaron el tamaño de la L2 en 8 veces.

Tal y como se puede observar en Figura 9 la cache L1 no comparte su espacio con la memoria compartida, si no con la cache de texturas.

Además mejoraron el rendimiento de las operaciones atómicas ejecutadas sobre la memoria compartida.

En definitiva GM107 no incluyó grandes cambios en la estructura de la micro-arquitectura, sino que simplemente han exprimido el rendimiento de la anterior micro-arquitectura.

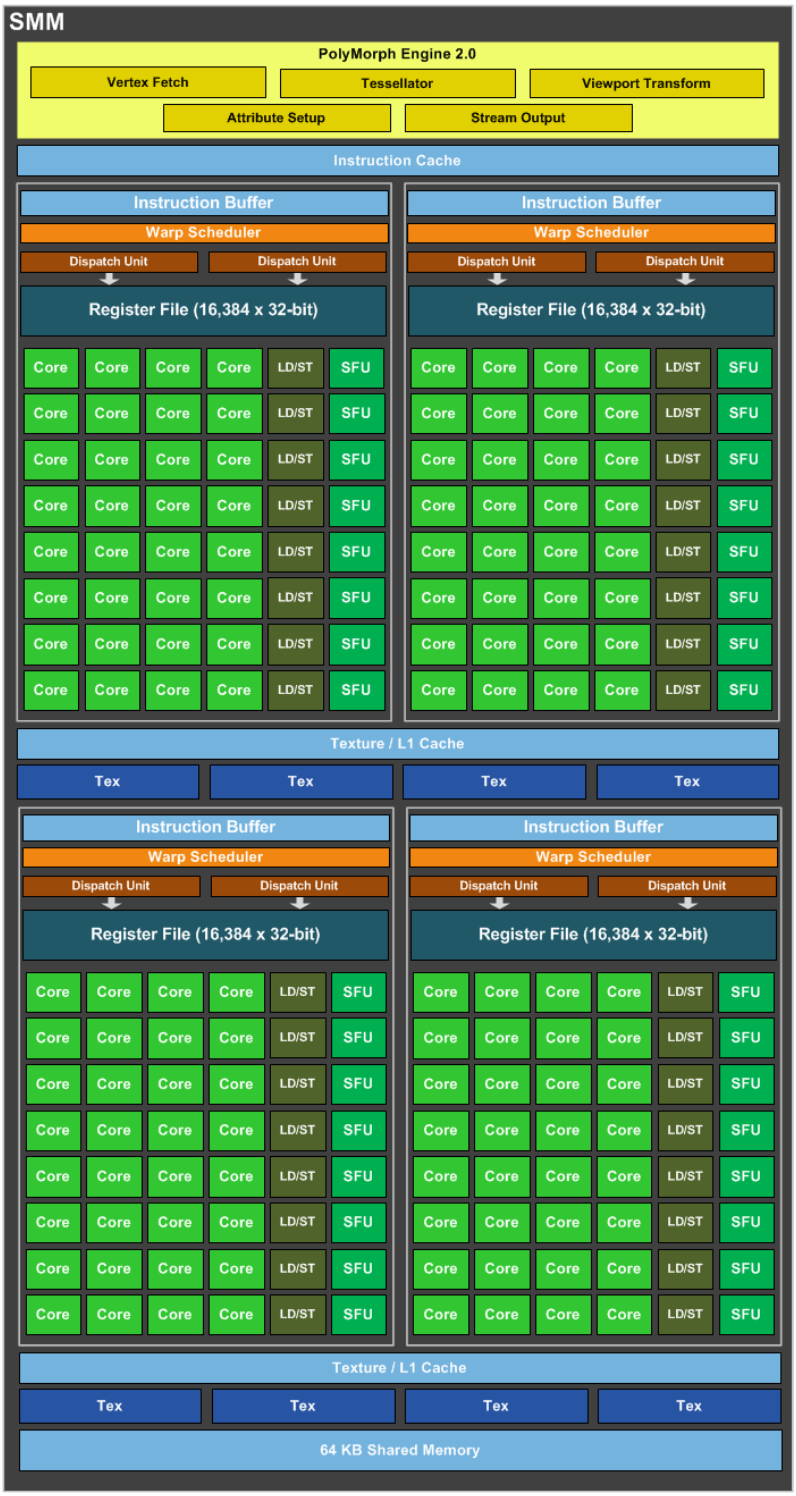

Figura 9. Multi-procesador de flujo de GM107

El nuevo chip GM204 no añadió cambios a la estructura de la micro-arquitectura, sin embargo consiguió mejorar aún más la eficiencia energética.

1.6.3 CUDA C/C++

CUDA es el lenguaje utilizado para desarrollar aplicaciones GPGPU en los procesadores gráficos NVIDIA, tal y como ya hemos mencionado antes.

En este apartado no describiré en lenguaje en su totalidad pues en cada versión del toolkit lo cambian y amplían, pero sí que describiré ciertos términos necesarios para la comprensión básica de cualquier aplicación desarrollada con CUDA.

Las aplicaciones que utilizan CUDA organizan el código GPGPU en núcleos. Una aplicación puede tener varios núcleos, que no son más que funciones de C que contienen código en CUDA.

Los núcleos son funciones que se ejecutan paralelamente por un número concreto de hilos.

La forma en que los hilos de un núcleo se organizan en rejilla. La distribución de esta viene determinada por la versión de lo que se conoce como CUDA capabilities (CC), que está directamente ligado con el hardware que ejecuta el núcleo.

La rejilla de los núcleos se dividen en bloques, y a su vez cada bloque tiene un número concreto de hilos. A parte internamente debido al hardware los hilos solo se pueden ejecutar en warps, que como ya se ha descrito antes son las unidades mínimas de ejecución, formada por un número concreto de hilos. Dicho número depende del CC del hardware que ejecute el núcleo.

En la se muestran las arquitecturas y GPUs con su respectiva versión de CUDA Capability.

Otro de los cambios importantes introducidos a CUDA fue la posibilidad de crear rejillas con formato 3D a partir de la versión CC 2.0. Es decir, a partir de esta versión se podían organizar los bloques en arrays tridimensionales, lo cual quitaba trabajo tanto al desarrollador como a los procesadores de flujo en determinados problemas.

Además, el paralelismo dinámico apareció como ya he explicado con anterioridad con el chip GK110, cuya versión CC es la 3.5.

Uno de los factores fundamentales para el desarrollo de núcleos eficientes es lo que se conoce como la ocupación CUDA. Esto es el uso teórico de los procesadores de flujo en tanto por ciento. Además de la ocupación teórica existe una ocupación práctica, la cual se obtiene con la aplicación de análisis de rendimiento de CUDA, y depende en su gran mayoría de la ocupación teórica, así como de los bloqueos que se den en la ejecución del código.

La ocupación teórica varía en función de la versión CC, el tamaño de la memoria compartida, el número de hilos por bloque, el número de registros por hilo, y el tamaño de la memoria compartida por bloque.

CC	Micro-arquitectura	GPUs
1.0	Tesla	G80
1.1		G92
		G94
		G96
		G98
		G84
		G86
1.2		GT218
		GT216
		GT215
1.3		GT200
		GT200b
2.0	Fermi	GF100
		GF110
2.1		GF104
		GF106
		GF108
		GF114
		GF116
		GF117
		GF119
3.0	Kepler	GK104
		GK106
		GK107
3.2		Tegra K1
3.5		GK110
		GK208
3.7		GK210
5.0	Maxwell	GM107
		GM108
5.2		GM200
		GM204
		GM206
5.3		Tegra X1

Tabla 1. Tabla con las versiones CC de las GPUs y micro-arquitecturas

Dada la gran complejidad de la ecuación para el cálculo de la ocupación teórica NVIDIA puso a disposición de los desarrolladores una hoja de cálculo (5) para obtener el valor de la ocupación a partir de los parámetros arriba mencionados.

Además de la hoja de cálculo en las últimas versiones de CUDA se ha añadido una librería para el cálculo de los parámetros óptimos para la ejecución.

El objetivo de los desarrolladores debería ser el de obtener una ocupación teórica lo más cercana posible al 100 %. Este es un factor que hay que tener en cuenta a la hora de diseñar el algoritmo para CUDA, y es por tanto imperante planificar con mucho cuidado el valor de los parámetros que influyen en la ocupación.

Una alta ocupación se traduce en un tiempo de ejecución óptimo. Aun así la ocupación teórica casi nunca coincide con la ocupación real. Los factores que influyen en la disminución de la ocupación real son la divergencia en la ejecución, la carga no correlativa de información desde memoria global, o las colisiones de acceso de escritura a la memoria compartida.

La divergencia es el fenómeno asociado a los bloques condicionales de un código. Debido a que los hilos se ejecutan en conjuntos discretos llamados warps, si en alguno de los hilos el camino de ejecución diverge de los demás, estos se bloquean hasta volver a sincronizarse, con lo cual se desaprovecha la capacidad de cómputo.

Respecto a la memoria global, cada multi-procesador de flujo almacena las direcciones de memoria requeridas y las carga en bloques. Es por esto que cuanto más consecutivas sean las direcciones de memoria menor será la información cargada y por tanto más pequeño será el tiempo empleado en la transferencia de memoria. Este factor es bastante importante, pues la latencia de la memoria global es bastante alta.

La memoria compartida se organiza en bancos, en cada acceso es posible realizar un acceso a cada banco de manera simultánea. Es por eso que se debe evitar en la medida de lo posible acceder a direcciones de memoria compartida que caigan en el mismo banco. Esto es, si una micro-arquitectura tuviera 32 bancos de memoria compartida, cada 32 posiciones de memoria se produciría una colisión y la transferencia de memoria se retrasaría un ciclo.

1.7 HERRAMIENTAS MATEMÁTICAS

Hay ciertos conceptos que se mocionarán en futuros apartados los cuales es necesario conocer para entender lo que se describe en dichos apartados. Además hay ciertos cálculos que también ayudarán en futuras secciones a la lectura de estas.

1.7.1 Conceptos geométricos relacionados

- **N-símplice**. Sea $\mathbb{R}^n$ el espacio vectorial euclídeo se define un n-símplice estándar como la envoltura convexa de $(n+1)$ puntos independientes.
- **Politopo**. Un politopo es todo aquel objeto geométrico con lados planos. Este concepto se usa como generalización de polígono y poliedro en dimensiones superiores, y se nombran como n-politopo, siendo n las dimensiones. Por ejemplo, varios 4-polítopos son el teseracto o el pentácoro. Es importante notar que todos los n-politopos están compuestos por varios (n-1)-politopos, inclusive los 0-politopos, los cuales están compuestos por politopos nulos.
- **Descomposición simplicial.** Una descomposición simplicial una descomposición en n-símplices de un n-politopo. Cada n-símplice de la descomposición estarán formados por los vértices de cada elemento del n-politopo, además de un vértice común previamente escogido.

1.7.2 Cálculo del n-Volumen

Una de las herramientas fundamentales en los algoritmos a describir será el n-Volumen, más concretamente el área de un triángulo y el volumen de un tetraedro.

Si se tiene un *n*-símplex con vértices $\{(x_{i1},\cdots,x_{in}), 1 \leq i \leq n+1\}$, el *n*-Volumen se calcula mediante la fórmula de Lagrange, descrita en la Ecuación 1.

$$V(Q) = \frac{1}{n!}\begin{vmatrix} x_{11} & x_{12} & \cdots & x_{1n} & 1 \\ x_{21} & x_{22} & \cdots & x_{2n} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{(n+1)1} & x_{(n+1)2} & \cdots & x_{(n+1)n} & 1 \end{vmatrix}$$

Ecuación 1. Fórmula para el cálculo del n-Volumen

Cuando $V(Q)$ es cero decimos que el n-símplex está degenerado, pues al menos uno de sus vértices es dependiente de alguno otro.

1.7.3 Cálculo del signo del símplice

Siendo Q el conjunto de los vértices del símplice, usando la fórmula del n-Volumen descrita en la Ecuación 1 podemos definir la función $sign(Q)$ tal y como se muestra en la Ecuación 2.

$$sign(Q) = \begin{cases} 1 & si\ V(Q) > 0 \\ 0 & si\ \ V(Q) = 0 \\ -1 & si\ V(Q) < 0 \end{cases}$$

Ecuación 2. Función de signo de un n-símplex

1.7.4 Coordenadas baricéntricas

Cada n-símplex puede definir su propio sistema de coordenadas baricéntricas, de tal modo cualquier punto puede ser expresado en función de las coordenadas de los vértices del n-símplex.

Esto es altamente útil para determinar si un punto se encuentra en el exterior o en el interior del n-símplex.

Cada componente de las coordenadas baricéntricas corresponde al n-Volumen de cada uno de los símplices de la descomposición simplicial usando como punto adicional el punto P, tal y como se ejemplifica en la Figura 10.

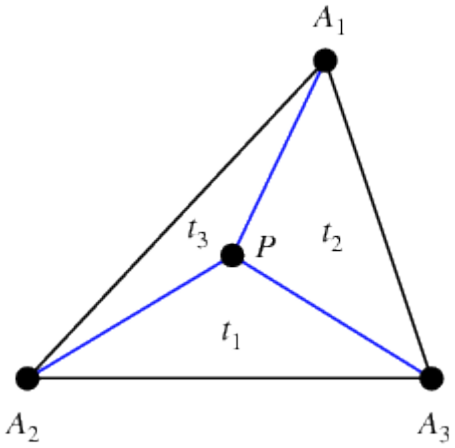

Figura 10. Coordenadas baricéntricas (t_1, t_2, t_3)

Es importante destacar que solo pueden definirse sistemas de coordenadas baricéntricas a partir de n-símplex no degenerados.

Para realizar el cambio de coordenadas baricéntricas a cartesianas es necesario definir la matriz de transformación de base.

Si se tiene un sistema de coordenadas baricéntricas formado por un *n*-símplex con vértices $\{v_0, \cdots, v_n\}$, su matriz de transformación de base es como la descrita en la Ecuación 3.

$$T = \begin{pmatrix} v_0 - v_n \\ \vdots \\ v_{n-1} - v_n \end{pmatrix}^T$$

Ecuación 3. Matriz de transformación de base

A partir de la matriz de transformación de base podemos escribir la transformación lineal para el cambio de base en forma algebraica, tal y como muestra la Ecuación 4.

$$T \cdot \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_{n-1} \end{pmatrix} + (v_n)^T = (P)^T$$

Ecuación 4. Transformación de cambio de coordenadas

Dónde: P es el punto en coordenadas cartesianas.

$(\lambda_0, \cdots, \lambda_n)$ es el punto en coordenadas baricéntricas.

La transformación inversa sería la plasmada en la Ecuación 5.

$$\begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_{n-1} \end{pmatrix} = T^{-1} \cdot (P - v_n)^T$$

$$\lambda_n = 1 - \sum_{i=0}^{n-1} \lambda_i$$

Ecuación 5. Transformación inversa de cambio de coordenadas

1.7.5 Test de inclusión de punto en símplice

Uno de los principales usos que tiene el cálculo del sistema de coordenadas baricéntricas de un n-símplex es el de poder determinar la posición relativa de un punto con respecto al símplice.

Siendo $\lambda = (\lambda_{0,\dots,}\lambda_n)$ las coordenadas baricéntricas del punto evaluado podemos decir que el punto se encuentra en el símplice si $\lambda_i > 0 \ \ \forall i \in [0, n]$.

El punto evaluado se encuentra fuera si $\exists i \in [0, n] \mid \lambda_i < 0$

1.8 PLANIFICACIÓN

En esta sección se pretende llevar a cabo una previsión de las diferentes fases necesarias para llevar a cabo el proyecto, así como una estimación de la duración de dichas fases, aun siendo un proyecto de tipo experimental.

Este proyecto se divide en los siguientes bloques claramente diferenciados:

- Estudio del problema tratado.
- Documentación sobre las arquitecturas tratadas.
- Diseño del nuevo algoritmo
- Implementación en las distintas arquitecturas.
- Obtención de las estadísticas.
- Redacción de la documentación.

Cabe mencionar, que durante el desarrollo del proyecto se han introducido una serie de hitos o puntos de control cuyo objetivo no es más que el de verificar la correcta evolución del proyecto a lo largo de su proceso de desarrollo.

Por otro lado es importante mencionar que para el proyecto se desarrollará en una jornada laboral de lunes a viernes a tiempo completo, empleando por tanto 8h por día invertido en el proyecto.

1.8.1 Descripción detallada de la planificación

En este apartado describo con detalle las tareas que conforman el proyecto.

1.8.1.1 Reunión con el tutor (N)

Durante estas fases del proyecto, me reúno con el tutor de mi proyecto con el objetivo de verificar el correcto progreso del mismo, así como para resolver cualquier duda que haya podido surgir durante las fases previas a dicha reunión.

Tal y como quizás se haya podido observar, desde el principio del proyecto se definió claramente una frecuencia de una reunión con el tutor cada 2 semanas, similar a la duración de un sprint en la metodología SCRUM. Además la duración de las reuniones no será superior a 2 horas con el objeto de evitar caer en redundancias.

1.8.1.2 Estudio y evaluación del proyecto

Con la información obtenida en la primera reunión con el tutor se llevará a cabo un desglose y revisión de la misma con el objetivo de obtener los conocimientos previos necesarios para la realización del proyecto, así como para obtener información de la rentabilidad de la realización del proyecto, teniendo en cuenta los riesgos y ventajas que pueden surgir como consecuencia de su puesta en marcha.

Se emplearán 3 días para el desarrollo de esta fase.

1.8.1.3 Búsqueda bibliográfica

Se trata de una fase basada en la búsqueda de información relacionada con el problema a tratar en el proyecto.

Se emplearán 6 días para el desarrollo de esta fase.

1.8.1.4 Revisión bibliográfica

En esta etapa se lleva a cabo un filtrado de información obtenida en la fase anterior.

Se emplearán 4 días para el desarrollo de esta fase.

1.8.1.5 Inicio de la documentación

Una vez en este punto se desarrollará la primera parte de la parte de la documentación que define la introducción a la misma, en la cual se reflejará el contenido de toda la investigación llevada a cabo hasta el momento.

Se emplearán 5 días para el desarrollo de esta fase.

1.8.1.6 Hito 1

En este punto se realizará una retrospectiva de los pasos llevados a cabo hasta el momento, intentado obtener cuales son los aspectos que se pueden mejorar, así como un repaso muy breve de diferentes aspectos que podamos haber pasado por alto. Sobre todo, esta tarea marca el final del proceso de recolección de información, para comenzar con el desarrollo de la solución.

1.8.1.7 Configuración del toolkit de CUDA

En esta fase se prepara el toolkit de CUDA, instalando la integración con Visual Studio, así como revisando alguno de los ejemplos ya existentes que se han desarrollado con CUDA.

Se empleará 1 día para el desarrollo de esta fase.

1.8.1.8 Elaboración de la versión 0 del algoritmo

Una vez configurado el toolkit de CUDA, en esta fase del proyecto se diseñará una primera versión del algoritmo que dará solución al problema planteado. Es posible que sea la versión definitiva del algoritmo, aunque dado que se trata de un proyecto de optimización, es poco probable.

Se emplearán 3 días para el desarrollo de esta fase.

1.8.1.9 Implementación monohilo de la versión 0 del algoritmo

En esta fase se trata de implementar la versión preliminar del algoritmo que se ha diseñado en la anterior fase del proceso.

Se emplearán 3 días para el desarrollo de esta fase.

1.8.1.10 Obtención parcial de estadísticas del algoritmo 0

Tras la implementación de la versión preliminar del algoritmo, se obtienen una serie de datos estadísticos con el objetivo de evaluar el rendimiento computacional de dicha versión.

Se emplearán 3 días para el desarrollo de esta fase.

1.8.1.11 Revisión de la versión 0 del algoritmo

Una vez obtenidos los datos necesarios de la versión preliminar del algoritmo, se realiza una evaluación de los mismos y se decide qué aspectos del mismo son necesarios modificar, para diseñar una posterior versión.

Se emplearán 6 días para el desarrollo de esta fase.

1.8.1.12 Elaboración de la versión definitiva del algoritmo

Tras evaluar aquellos aspectos a mejorar de la versión preliminar del algoritmo, se realiza una nueva versión del mismo.

Si fuera necesario, al final del proceso de evaluación de este algoritmo, se debería volver a este punto, con el objetivo de mejorarlo, hasta obtener un resultado razonable. Esto por supuesto, supondría un retraso en el proyecto.

Se emplearán 6 días para el desarrollo de esta fase.

1.8.1.13 Implementación monohilo de la versión definitiva

En esta fase se trata de implementar la versión monohilo definitiva del algoritmo que se ha diseñado en la anterior fase del proceso.

Se emplearán 4 días para el desarrollo de esta fase.

1.8.1.14 Implementación multi-hilo de la versión definitiva

En esta fase se trata de implementar la versión multi-hilo definitiva del algoritmo que se ha diseñado en la anterior fase del proceso.

Se emplearán 5 días para el desarrollo de esta fase.

1.8.1.15 Implementación CUDA de la versión definitiva

En esta fase se trata de implementar la versión CUDA definitiva del algoritmo que se ha diseñado en la anterior fase del proceso.

Se emplearán 6 días para el desarrollo de esta fase.

1.8.1.16 Obtención parcial de estadísticas del algoritmo definitivo

Tras la implementación de la versión definitiva del algoritmo, se obtienen una serie de datos estadísticos con el objetivo de evaluar el rendimiento computacional de dicha versión.

Se emplearán 3 días para el desarrollo de esta fase.

1.8.1.17 Revisión de la versión definitiva del algoritmo

Una vez obtenidos los datos necesarios de la versión definitiva del algoritmo, se realiza una evaluación de los mismos y se decide si es necesario modificar algún aspecto de esta versión. En el caso de tener que modificar algún aspecto se debería rediseñar el algoritmo.

Se emplearán 6 días para el desarrollo de esta fase.

1.8.1.18 Desarrollo de una aplicación práctica para el algoritmo

Una vez desarrollada e implementada la versión definitiva del algoritmo, se pasa a desarrollar una aplicación práctica del algoritmo, con el objetivo de demostrar la efectividad del mismo.

En este caso, la aplicación en cuestión, es la de voxelización de la un modelo CSG cuyas primitivas son representaciones B-Rep.

Se emplearán 7 días para el desarrollo de esta fase.

1.8.1.19 Obtención de los datos necesarios para la ejecución de la aplicación práctica del algoritmo

Para poder ejecutar la aplicación práctica desarrollada es necesario modelar/obtener los modelos CSG.

Se emplearán 2 días para el desarrollo de esta fase.

1.8.1.20 Completar la memoria del proyecto (N)

Al final de los hitos 2 y 3 es necesario seguir completando el boceto de la documentación con aquellos aspectos relevantes tratados en estos hitos.

Se emplearán 9 días para el desarrollo de esta fase, aunque no de manera secuencial, sino más bien de manera paralela al desarrollo del hito.

1.8.1.21 Hito 2

En este punto se realizará una retrospectiva de los pasos llevados a cabo hasta el momento, intentado obtener cuales son los aspectos que se pueden mejorar, así como un repaso muy breve de diferentes aspectos que podamos haber pasado por alto. Sobre todo, esta tarea marca el final del proceso de desarrollo de la solución, para comenzar con el proceso de recolección de estadísticas.

1.8.1.22 Búsqueda de los modelos de prueba

Antes de obtener las estadísticas finales para el algoritmo definitivo es necesario hacerse de unos modelos representativos para realizar las pruebas. No solo estos han de ser representativos, sino además hay que topológica y geométricamente correctos.

Se empleará 1 día para el desarrollo de esta fase.

1.8.1.23 Obtención de las estadísticas para la versión monohilo del algoritmo

En esta fase se obtienen las estadísticas finales para la versión monohilo del algoritmo definitivo.

Se emplearán 9 días para el desarrollo de esta fase.

1.8.1.24 Obtención de las estadísticas para la versión multi-hilo del algoritmo

En esta fase se obtienen las estadísticas finales para la versión multi-hilo del algoritmo definitivo.

Se emplearán 9 días para el desarrollo de esta fase.

1.8.1.25 Obtención de las estadísticas para la versión CUDA del algoritmo

En esta fase se obtienen las estadísticas finales para la versión CUDA del algoritmo definitivo.

Se emplearán 6 días para el desarrollo de esta fase.

1.8.1.26 Generación de las gráficas de las estadísticas

En esta fase se generan las gráficas asociadas a las estadísticas de las versiones monohilo, multi-hilo y CUDA para la versión definitiva del algoritmo.

Se emplearán 3 días para el desarrollo de esta fase.

1.8.1.27 Hito 3

En este punto se realizará una retrospectiva de los pasos llevados a cabo hasta el momento, intentado obtener cuales son los aspectos que se pueden mejorar, así como un repaso muy breve de diferentes aspectos que podamos haber pasado por alto. Sobre todo, esta tarea marca el final del proceso de recolección de estadísticas, para comenzar la parte final del proyecto, en la cual se revisan los últimos detalles del mismo.

1.8.1.28 Terminar la memoria del proyecto

En esta fase se llevará a cabo el completado de toda la información que conforme la documentación del proyecto, teniendo en cuenta las mejoras posibles que sean viables introducir en función provenientes de la fase de pruebas y de la información obtenida en las retrospectivas correspondientes a cada uno de los hitos.

1.8.1.29 Preparar el entregable del proyecto

En esta fase del proyecto se preparará los resultados a entregar con el proyecto, tales como el código fuente, la memoria, los modelos y la documentación adicional.

Se empleará 1 día para el desarrollo de esta fase.

1.8.1.30 Revisión final del proyecto

En este punto llevaremos a cabo una revisión general del proyecto, con el fin de comprobar el correcto funcionamiento del sistema y de poder validar la integridad y la correcta disposición de esta memoria.

También se llevará a cabo la preparación de la presentación del proyecto, con la consiguiente reunión con los tutores para obtener el visto bueno del mismo su parte.

Se emplearán 9 días en la realización de esta tarea.

1.8.2 Distribución temporal del proyecto

Para mostrar de la forma más clara posible la organización temporal de las fases anteriormente indicadas dentro de los plazos de tiempo se muestra la Tabla 2.

Tarea	Nombre de tarea	Duración	Comienzo	Fin	Anteriores
1	Inicio del proyecto	0 días	mar 10/02/15	mar 10/02/15	
2	Reunión con el tutor (1)	0,25 días	mar 10/02/15	mar 10/02/15	1
3	Estudio y evaluación del proyecto	3 días	mié 11/02/15	vie 13/02/15	2
4	Búsqueda bibliográfica	6 días	lun 16/02/15	lun 23/02/15	3
5	Reunión con el tutor (2)	0,25 días	mar 24/02/15	mar 24/02/15	4
6	Revisión bibliográfica	4 días	mié 25/02/15	lun 02/03/15	5
7	Inicio de la documentación	5 días	mar 03/03/15	lun 09/03/15	6
8	Reunión con el tutor (3)	0,25 días	mar 10/03/15	mar 10/03/15	7
9	Hito 1	0 días	mié 11/03/15	mié 11/03/15	8
10	Configuración del toolkit de CUDA	1 día	mié 11/03/15	mié 11/03/15	9
11	Elaboración de la versión 0 del algoritmo	3 días	jue 12/03/15	lun 16/03/15	10
12	Implementación monohilo de la versión 0 del algoritmo	4 días	mar 17/03/15	vie 20/03/15	11
13	Implementación CUDA de la versión 0 del algoritmo	5 días	mar 17/03/15	lun 23/03/15	11
14	Reunión con el tutor (4)	0,25 días	mar 24/03/15	mar 24/03/15	12;13
15	Obtención parcial de estadísticas del algoritmo 0	3 días	mié 25/03/15	vie 27/03/15	14

16	Revisión de la versión 0 del algoritmo	6 días	lun 30/03/15	lun 06/04/15	15
17	Reunión con el tutor (5)	0,25 días	mar 07/04/15	mar 07/04/15	16
18	Elaboración de la versión definitiva del algoritmo	3 días	mié 08/04/15	vie 10/04/15	17
19	Implementación monohilo de la versión definitiva	4 días	lun 13/04/15	jue 16/04/15	18
20	Implementación multi-hilo de la versión definitiva	5 días	lun 13/04/15	vie 17/04/15	18
21	Implementación CUDA de la versión definitiva	6 días	lun 13/04/15	lun 20/04/15	18
22	Reunión con el tutor (6)	0,25 días	mar 21/04/15	mar 21/04/15	19;20;21
23	Obtención parcial de estadísticas del algoritmo definitivo	3 días	mié 22/04/15	vie 24/04/15	22
24	Revisión de la versión definitiva del algoritmo	6 días	lun 27/04/15	lun 04/05/15	23
25	Reunión con el tutor (7)	0,25 días	mar 05/05/15	mar 05/05/15	24
26	Desarrollo de una aplicación práctica para el algoritmo	7 días	mié 06/05/15	jue 14/05/15	25
27	Obtención de los datos necesarios para la ejecución de la aplicación práctica del algoritmo	2 días	vie 15/05/15	lun 18/05/15	26
28	Completar la memoria del proyecto (1)	9 días	mié 11/03/15	lun 23/03/15	9
29	Reunión con el tutor (8)	0,25 días	mar 19/05/15	mar 19/05/15	28;27
30	Hito 2	0 días	mié 20/05/15	mié 20/05/15	29
31	Búsqueda de los modelos de prueba	1 día	mié 20/05/15	mié 20/05/15	30
32	Obtención de las estadísticas para la versión monohilo del algoritmo	9 días	mié 20/05/15	lun 01/06/15	30
33	Reunión con el tutor (9)	0,25 días	mar 02/06/15	mar 02/06/15	32;31
34	Obtención de las estadísticas para la versión multi-hilo del algoritmo	9 días	mié 03/06/15	lun 15/06/15	33
35	Reunión con el tutor (10)	0,25 días	mar 16/06/15	mar 16/06/15	34
36	Obtención de las estadísticas para la versión CUDA del algoritmo	6 días	mié 17/06/15	mié 24/06/15	35

37	Generación de las gráficas de las estadísticas	3 días	jue 25/06/15	lun 29/06/15	36
38	Completar la memoria del proyecto (2)	9 días	mié 20/05/15	lun 01/06/15	30
39	Reunión con el tutor (11)	0,25 días	mar 30/06/15	mar 30/06/15	37;38
40	Hito 3	0 días	mié 01/07/15	mié 01/07/15	39
41	Terminar la memoria del proyecto	8 días	mié 01/07/15	vie 10/07/15	40
42	Preparar el entregable del proyecto	1 día	lun 13/07/15	lun 13/07/15	41
43	Reunión con el tutor (12)	0,25 días	mar 14/07/15	mar 14/07/15	42
44	Revisión final del proyecto	9 días	mié 15/07/15	lun 27/07/15	43
45	Reunión con el tutor (13)	0,25 días	mar 28/07/15	mar 28/07/15	44
46	Fin del proyecto	0 días	mar 28/07/15	mar 28/07/15	45

Tabla 2. Distribución temporal del proyecto

A continuación, en la Figura 11, se muestra una gráfica con el número de días invertidos para cada tarea.

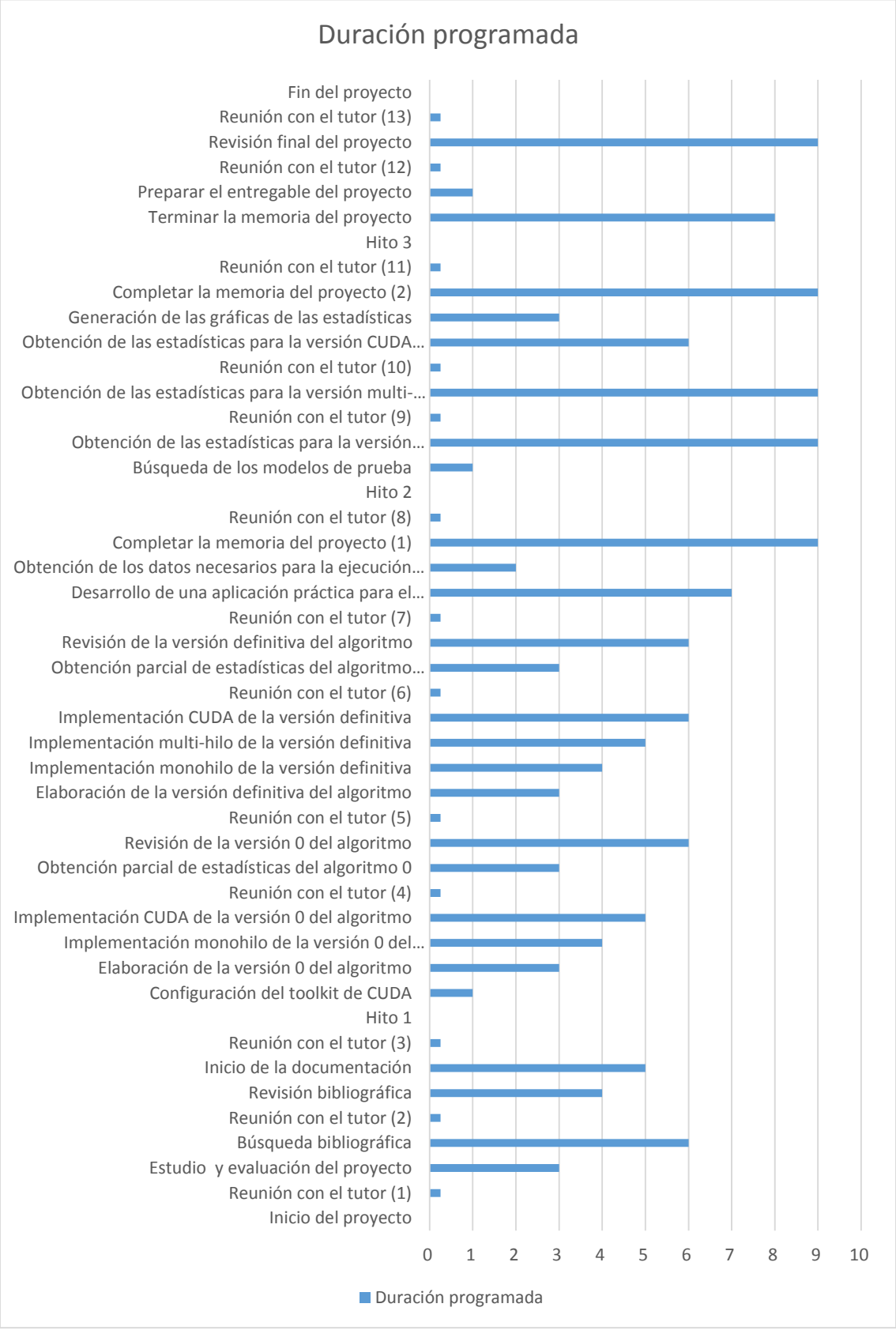

Figura 11. Duración de las tareas del proyecto en días

1.8.3 Diagrama de Gantt

En la Figura 12, Figura 13, Figura 14, Figura 15, Figura 16 y la Figura 17 se muestra el diagrama de Gantt que presenta la conexión entre cada una de las tareas, con sus fechas de inicio y de finalización.

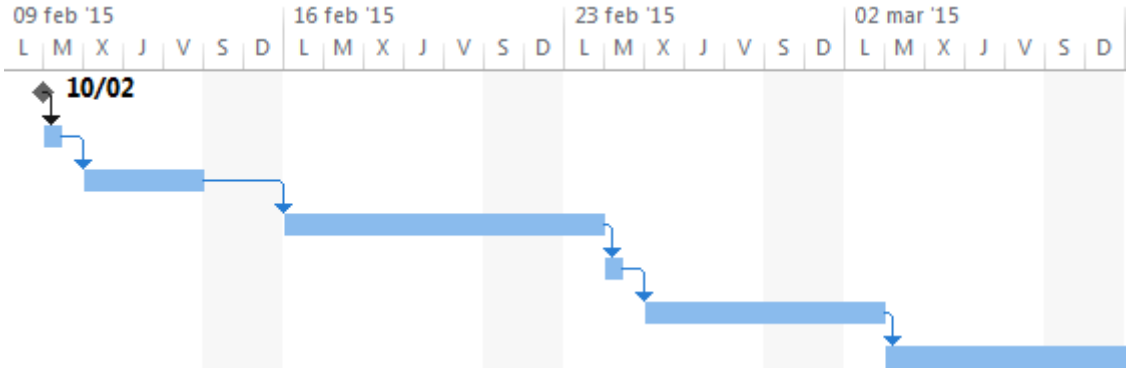

Figura 12. Diagrama de Gantt 09/02/15 – 09/03/15

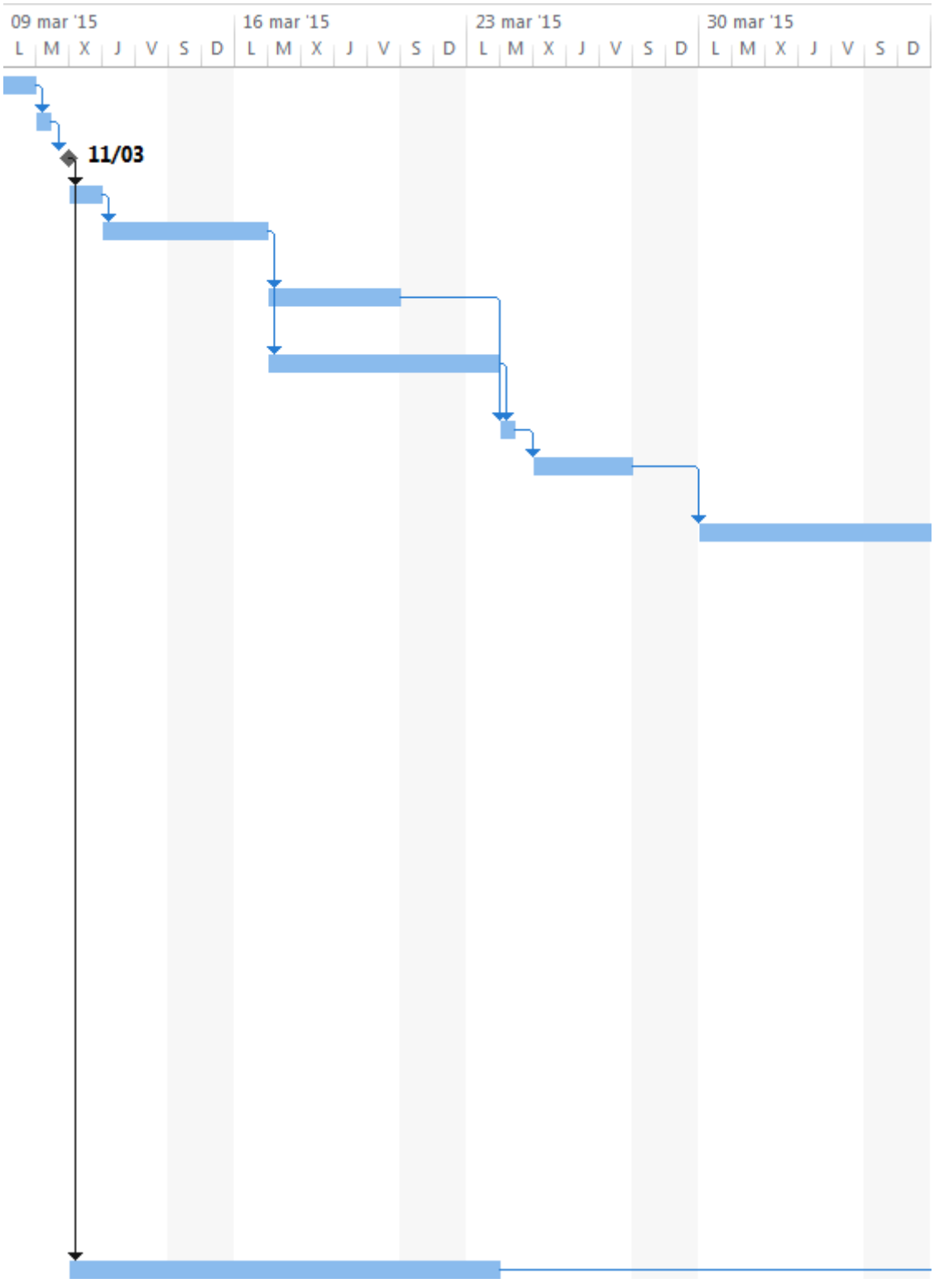

Figura 13. Diagrama de Gantt 09/03/15 – 06/04/15

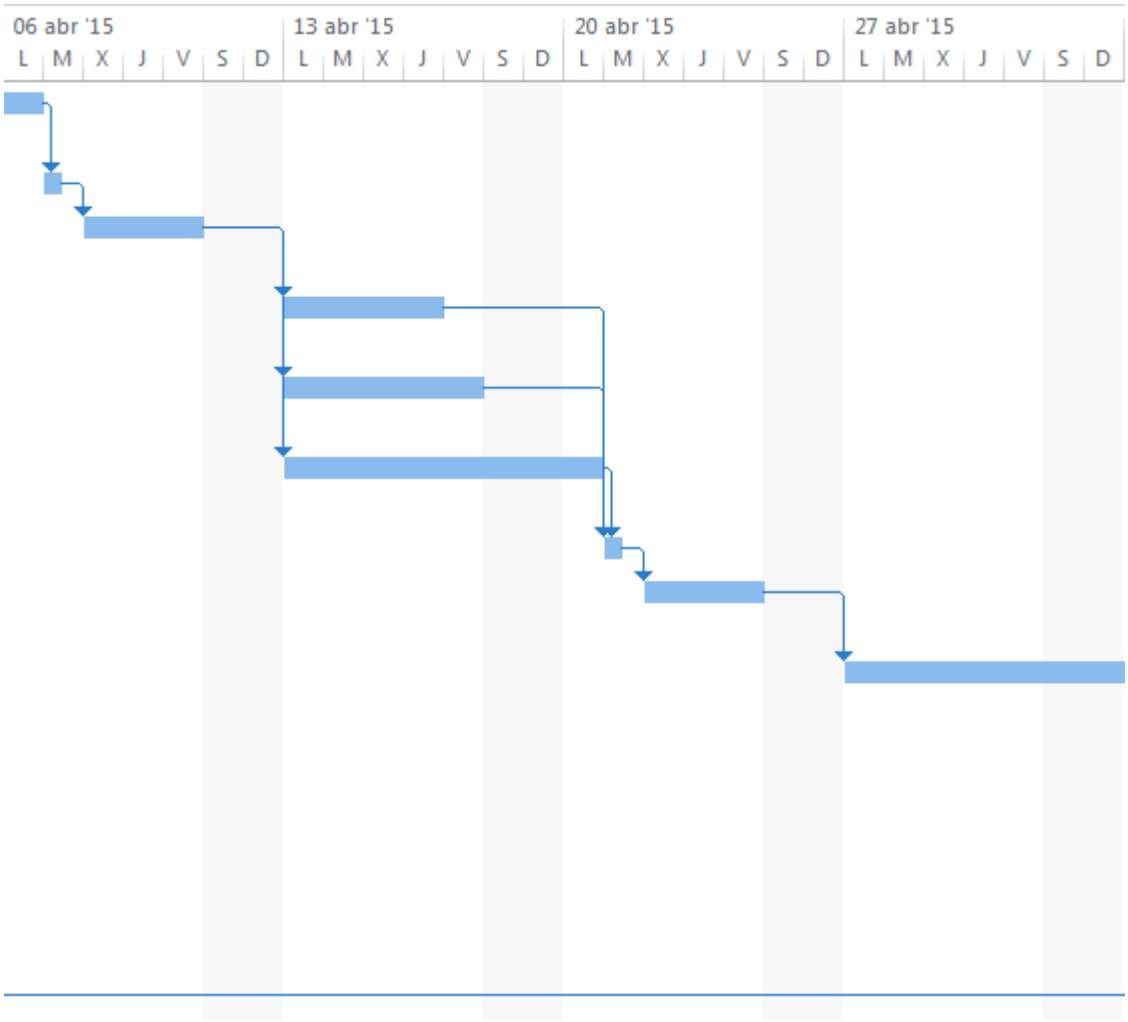

Figura 14. Diagrama de Gantt 06/04/15 – 04/05/15

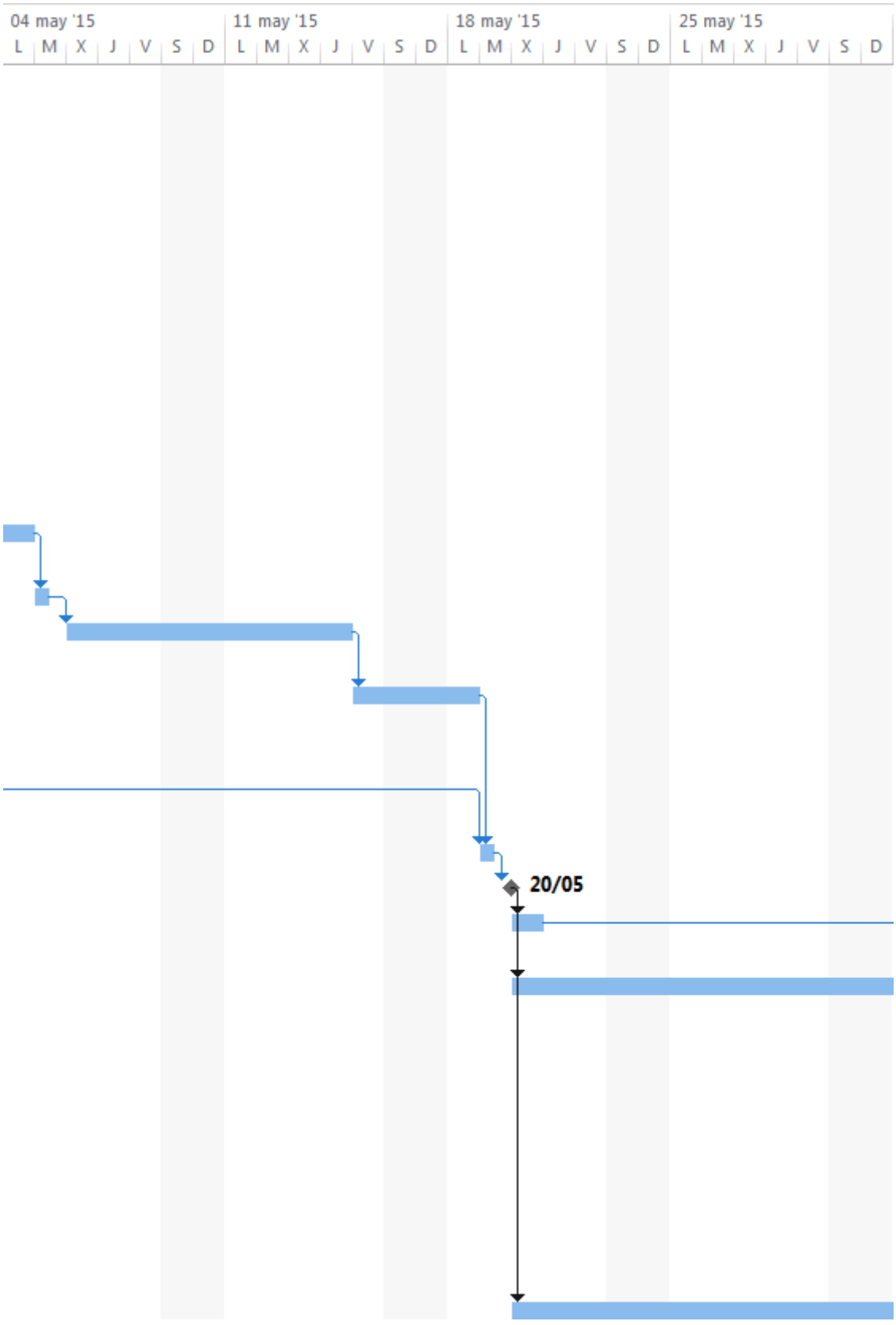

Figura 15. Diagrama de Gantt 04/05/15 – 01/06/15

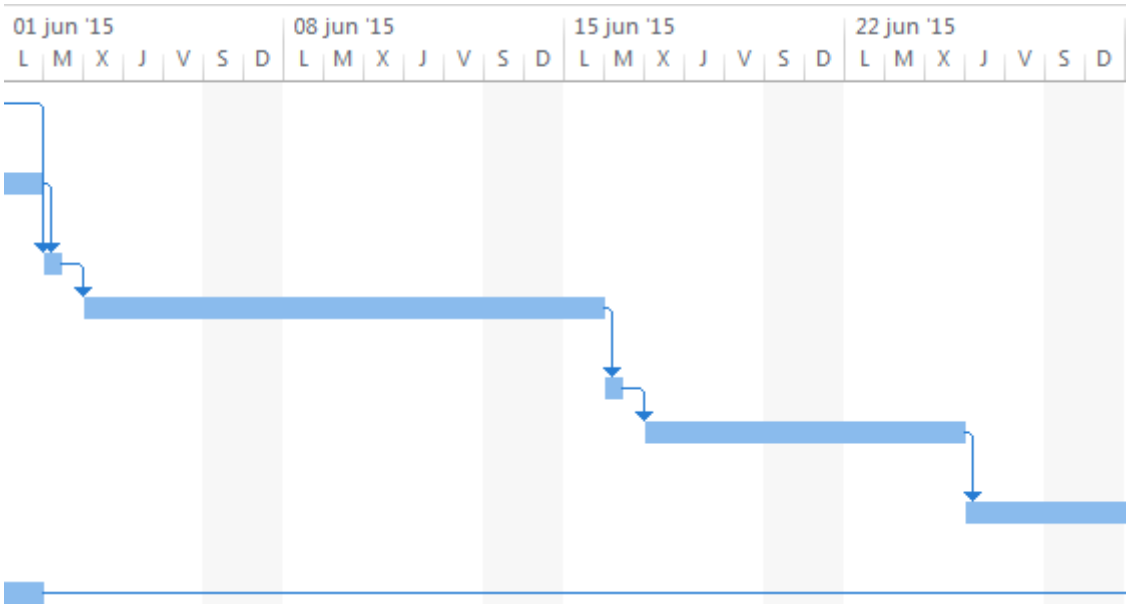

Figura 16. Diagrama de Gantt 01/06/15 – 29/06/15

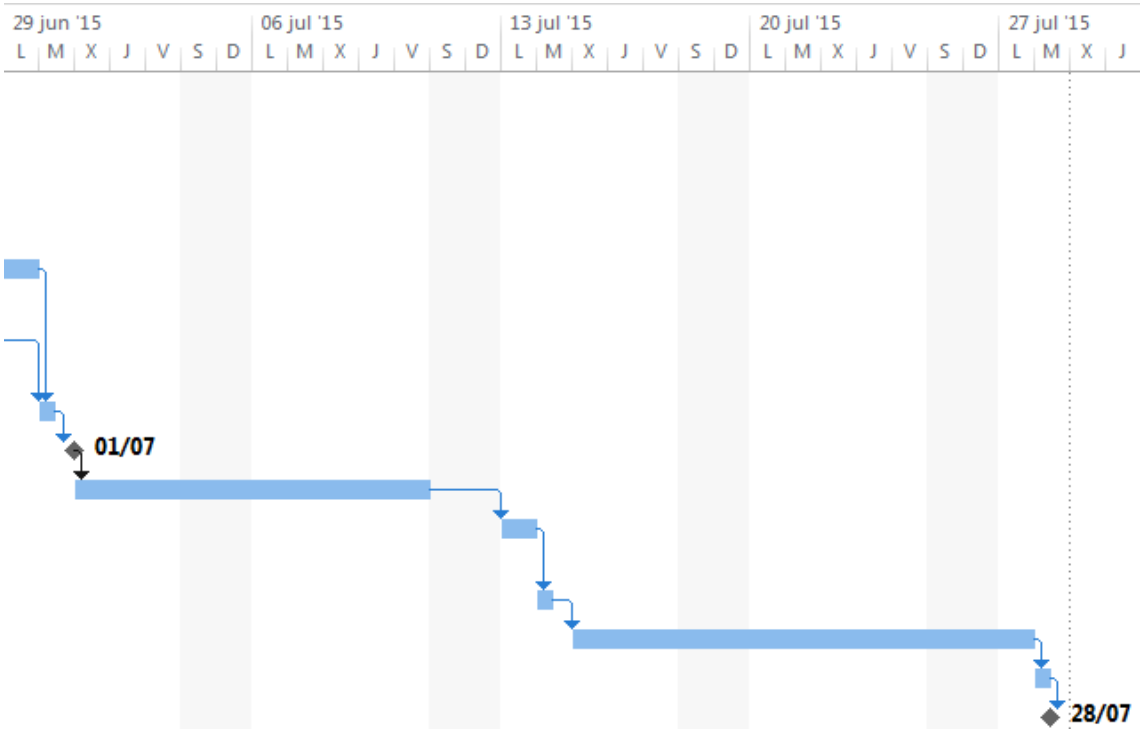

Figura 17. Diagrama de Gantt 29/06/15 – 30/07/15

2 ESTADO DEL ARTE

2.1 INTRODUCCIÓN

Es importante repasar los antecedentes del problema en el que se centra el proyecto, que es la resolución de la inclusión de punto en politopo. De este modo se puede comprender los pasos realizados en la resolución del problema para explicar el diseño del nuevo algoritmo presentado en este proyecto.

Además también es fundamental evaluar el uso de otro tipo de hardware al empleado en el proyecto (GPU).

2.2 ESTADO DEL ARTE DE LOS ALGORITMOS

Muchos han sido los autores que han presentado algoritmos para la resolución del problema que tratamos. Algunos han llegado a una solución totalmente valida, otros no. Incluso de los que han llegado a una solución válida, hay quienes han presentado algoritmos con implementaciones más rápidas que otros.

2.2.1 Algoritmo del número de intersecciones

Este algoritmo se basa en un teorema (6) presentado por el matemático francés Camille Jordan en 1887, el cual dice que todo bucle continuo sin intersecciones propias (conocido en geometría como curva cerrada) divide el espacio en el que se encuentra en dos partes, el interior y el exterior.

Fue M. Shimrat en 1962 el primero en presentar un algoritmo (7) basado en el anterior descrito teorema de Jordan.

Este autor propuso un algoritmo el cual consistía en lanzar un rayo con dirección arbitraria a partir del punto a evaluar e intersectarlo con el polígono. Tras esto se contaba el número de intersecciones, y tal como se observa en la Figura 1 si el número de intersecciones resulta ser par, se puede decir que el punto está afuera, en caso contrario el punto se encuentra dentro del polígono.

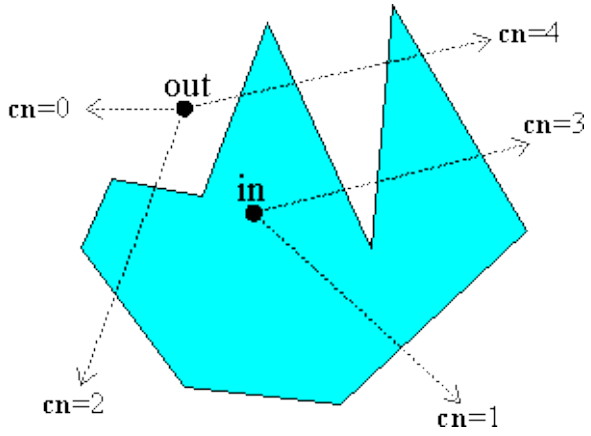

Figura 18. Demostración del algoritmo del número de intersecciones

El algoritmo es válido, sin embargo presenta determinados problemas pues las casuísticas crecen con el número de dimensiones.

En 2D, por ejemplo, si el rayo intersecta con un vértice del polígono no podemos determinar si contar una o dos intersecciones. Además si el rayo es linealmente dependiente con alguna de las aristas del polígono no puedes determinar el número de intersecciones pues en teoría intersectan en infinitos puntos.

Aunque se trata del algoritmo con menor fiabilidad de los que tratan de resolver el problema, bien es cierto que se trata del algoritmo más fácil de entender e implementar, además de que es fácilmente extensible a 3D.

Muchos desarrolladores optan por implementar este algoritmo lanzando un número de rayos impares con direcciones todas distintas entre ellos. Tras esto se realiza un análisis estadístico para determinar cuál es la opción más probable, bien que el punto esté dentro o que el punto se encuentre fuera.

Como último detalle cabe mencionar que este algoritmo no controla por defecto el caso en el que el punto se encuentre en la misma frontera que define el polígono, a diferencia de muchos otros algoritmos que si lo hacen.

2.2.2 Test de inclusión para polígonos

Este algoritmo fue presentado en un artículo escrito por F. Feito, J.C. Torres y A. Ureña (8). En el mismo artículo se presenta una evidencia que soporta al algoritmo.

Dicha evidencia dice que dado un polígono simple cuyos vértices tengan una correcta orientación se define una descomposición simplicial en triángulos originales, tal y como se muestra en la Figura 19.

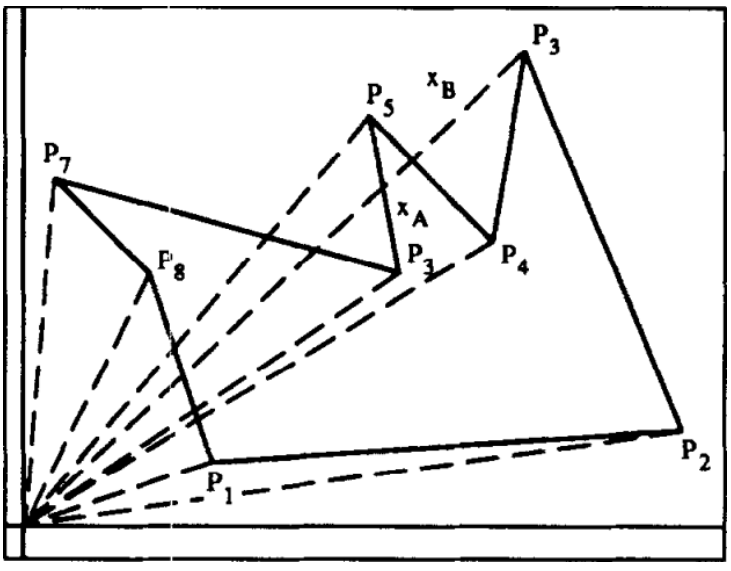

Figura 19. Descomposición en triángulos originales

Los triángulos originales no son más que triángulos formados por las aristas del polígono y un punto elegido arbitrariamente (preferiblemente fuera del AABB del polígono para evitar tener que manejar casuísticas adicionales).

Una vez tenemos los triángulos originales, se calcula el signo de estos, así como la inclusión del punto a evaluar en cada uno de estos triángulos originales.

Tras esto se genera un vector con tantos valores como aristas formen el polígono en el que cada componente de este corresponde al valor del signo del triángulo original en el caso de que el punto a evaluar se encuentre dentro, o cero en caso de que el punto evaluado se encuentre fuera.

Posteriormente se calcula el valor de inclusión, que no es más que la sumatoria de los componentes del anterior vector. Si el valor es positivo, el punto está dentro del polígono, si es negativo, el punto está fuera del polígono, y si es cero se encuentra en la misma frontera del polígono.

2.2.3 Test de inclusión para poliedros genéricos

Basándose en el anterior artículo (8), F. Feito y J. C. Torres publicaron un nuevo artículo (9) en el que adaptaban el algoritmo para 3D, el cual luego fue usado por J. Ruiz de Miras en la inclusión de punto en politopo (3).

Esta vez se descompone el poliedro en tetraedros originales.

Este algoritmo establecía los siguientes conceptos:

- $V^+(Q)$. Se refiere al conjunto de vértices que forman las aristas originales que bien pertenece a tetraedros con signo positivo o a vértices iniciales o finales de las caras positivas.
- $V^-(Q)$. Se refiere al conjunto de vértices que forman las aristas originales que bien pertenece a tetraedros con signo negativo o a vértices iniciales o finales de las caras negativas.
- $T_{or}\ (OV_1V_2V_3)$. Es el conjunto de caras del tetraedro original definido por el origen O y los vértices V_1, V_2, V_3.
- $Int(P)$. Es el interior de P en la topología correspondiente a la dimensión de P.
- $card(S)$. Refiriéndose a la cardinalidad del conjunto S.

Una vez definidos los anteriores conceptos, supongamos que tenemos un poliedro $P = \{F_1, F_2, \dots F_n\}$, siendo $F_i\ \forall i\ \in [1, n]$ las caras del poliedro definidas de la siguiente manera $F_i = \{V_{i,1}, V_{i,2}, \dots V_{i,e_i}\}$, siendo además e_i el número de vértices de la cara i.

Dicho esto en la definimos si un punto Q está en el interior del poliedro, teniendo en cuenta que ya que no se considera la frontera, si no se cumple lo descrito en la Ecuación 6. Descripción de la inclusión del punto en poliedro, entonces el punto Q está fuera del poliedro.

$$\mathrm{Q} \in Int(P) \Leftrightarrow \sum_{i=1}^{n}\sum_{j=1}^{e_i}\lambda_{ij} + card\left(V^{+}(Q)\right) - card\left(V^{-}(Q)\right) = 1$$

Donde $\lambda_{ij} =$

$$= \begin{cases} \mathrm{sign}\left(\mathrm{OV_{i,1}V_{i,j}V_{i,j+1}}\right) \text{ si } \mathrm{Q} \in \mathrm{Int}\left(\mathrm{OV_{i,1}V_{i,j}V_{i,j+1}}\right), 1 < \mathrm{j} < \mathrm{e_i} \\ \frac{1}{2}\mathrm{sign}\left(\mathrm{OV_{i,1}V_{i,j}V_{i,j+1}}\right) \text{ si } \mathrm{Q} \in \mathrm{Int}(\mathrm{T_k}),\ \text{con } \mathrm{T_k} \in \mathrm{T_{or}}(\mathrm{OV_{i,1}V_{i,j}V_{i,j+1}})\ 1 < \mathrm{j} < \mathrm{e_i} \\ 0 \text{ en cualquier otro caso} \end{cases}$$

Ecuación 6. Descripción de la inclusión del punto en poliedro

En la Figura 20 se muestra el pseudocódigo de algoritmo presentado en el artículo.

```
INCLUSION=0
V+ (Q)=empty
V- (Q)=empty
for i=1 to n {
        if (Q ∈ Fi) { INSIDE=true; EXIT }
        if ((Q∈ Int(OVi,1) and ((sign(OFi)>0 and not Vi,1 ∈ V+ (Q) or
          (sign(OFi)<0 and not Vi,1 ∈ V- (Q)) {
            INCLUSION=INCLUSION+sign(OFi)
            if (sign(OFi)<0)
                    V- (Q)= V- (Q) + Vi,1
            else    V+ (Q)= V+ (Q)+ Vi,1 }

        else if ((Q ∈ Int(OVi,ei) and ((sign(OFi) >0 and not Vi,ei ∈ V+ (Q) or
           (sign(OFi)<0 and not Vi,ei ∈ V- (Q) ))) {
            INCLUSIONS=INCLUSION+sign(OFi)
            if (sign(OFi) <0)
                    V-(Q)= V-(Q) + Vi,ei
            else    V+(Q)= V+(Q) + Vi,ei }
        else for j=2 to ei-1 {
            if ((Q ∈ Int (O,Vi,1, Vi,j) or (Q ∈ Int (O,Vi,j,Vi,j+1)
              or (Q ∈ Int (O,Vi,j+1,Vi,1)) {
                INCLUSION=INCLUSION + 1/2*sign([O,Vi,1,Vi,j,Vj+1])
            else if ((Q ∈ Int (OV i,j) and ( ( sign([O,Vi,1,Vi,j,Vj+1])>0
                and not Vi,j∈ V+ (Q) ) or (sign([O,Vi,1,Vi,j,Vi,j+1])<0
                and not Vi,j∈ V- (Q) ) {
                INCLUSION=INCLUSION+sign([O,Vi,1,Vi,j,Vi,j+1]
                if sign([O,Vi,1,Vi,j,Vj+1])<0
                    V- (Q)= V- (Q) + Vi,j
                else V+ (Q)= V+ (Q)+ Vi,j }
            else if (Q ∈ Int (O,Vi,1,Vi,j,Vi,j+1 )
                    INCLUSION=INCLUSION+sign([O,Vi,1,Vi,j,Vi,j+1])
        }
}

Let P=F1,F2,..,Fn be a polyhedron, where Fi (i=1..n) are the faces, and the vertex of
face i are: Fi=Vi,1,Vi,2,...,Vi,ei . Sign(OFi) is the sign of the pyramid OFi. The algorithm
decides if the point Q is inside the polyhedron, the result is obtained in the variable
INCLUSION, which will be 1 only if Q∈P.
```

Figura 20. Pseudocódigo del algoritmo de inclusión.

2.3 ESTADO DEL ARTE DEL HARDWARE

El uso del paradigma GPGPU no es la única alternativa para la computación geométrica intensiva.

Desde un principio las computaciones geométricas se realizaban en el coprocesador vectorial incluido con las CPUs, y ha sido una alternativa bastante buena. Aun así los coprocesadores han ido avanzando añadiendo cada vez más estándares más potentes.

Al respecto de los estándares de procesamiento vectorial para CPU, en los últimos años se han incorporado estándares como MMX, SSE2, SSE3, SSE4 AVX, AVX2, AVX-512, FMA.

Los anteriores mentados estándares permiten no solo trabajar con varios datos en una sola instrucción, sino además permiten trabajar con tipos de datos más grandes de 64-bits, lo cual es una interesante ventaja en entornos donde la precisión es crucial.

Otro enfoque es el seguido por Intel. Esta compañía ha optado por el desarrollo de tarjetas para la ayuda a la computación, que no son más que coprocesadores con varios núcleos integrados en un hardware con interfaz PCI-E. El producto al que me refiero que Intel viene desarrollando durante los últimos años se trata de los procesadores Intel Xeon Phi, los cuales pueden ofrecer una potencia de hasta 3 Tflops, por lo que suponen una gran competencia para la potencia presentada por los chips gráficos.

Uno de los últimos enfoque a tratar es el tradicional, el usado en los clústeres HPC clásicos, que no se trata más que de un clúster de CPUs multi-núcleo.

Dicho enfoque a mi parecer esta por extinguirse pues la eficiencia energética, lo cual es un factor importante a tratar cuando se trata de clústeres, se aleja en gran medida de soluciones GPGPU o de tarjetas aceleradoras.

Respecto al hardware de procesamiento, una de las tendencias actuales es la de aumentar la velocidad de ciclo. Y es que es esto lo que precisamente están estudiando en la DARPA (10), en un proyecto donde ya han conseguido fabricar transistores que alcanzan frecuencias del orden de 1 Thz.

Los avances producidos por la DARPA son gracias a que el nuevo transistor desarrollado es de un material diferente a los tradicionales transistores con base de silicio.

3 EL ALGORITMO

3.1 INTRODUCCIÓN

En esta sección se describe el nuevo algoritmo optimizado e implementado para CUDA, así como el proceso realizado para su desarrollo.

Empezaré por la descripción del proceso seguido durante el desarrollo del algoritmo, seguiré por la presentación del algoritmo auxiliar de la obtención de la posición relativa de un punto con respecto a un 3-símplex, para terminar con las distintas implementaciones del nuevo algoritmo.

3.2 PROCESO

Uno de los primeros pasos seguidos para el desarrollo del algoritmo fue el análisis de los algoritmos existentes que ya se han descrito en secciones previas.

Tras esto, se estudió los puntos clave de CUDA, para así poder desarrollar un algoritmo lo mejor optimizado posible.

Analizado, el problema, los demás algoritmos y las herramientas que CUDA proporcionaba, así como la manera de utilizarlas de la manera más óptima pasé a crear la nueva versión del algoritmo.

Al final la decisión que tomé para el desarrollo del algoritmo fue la de basarme en el algoritmo presentado por F. Feito y J.C. Torres (9), dado que este algoritmo presentaba un alto potencial para su adaptación al paradigma GPGPU.

El objetivo principal a la hora de diseñar este nuevo algoritmo era el de obtener una implementación sin divergencia, ya que como en anteriores secciones se ha explicado la existencia de divergencia en CUDA supone una disminución de la ocupación y en consecuencia un aumento en el tiempo de ejecución.

En una primera versión, el algoritmo construido no era robusto, aunque sin embargo únicamente aplicaba operaciones matemáticas, lógicas y de desplazamiento, lo cual suponía que no hubiera divergencia.

Ya que la versión del algoritmo descrita con anterioridad no era robusta, esta fue descartad, sin embargo, su diseño supuso un paso hacia lo que sería la primera versión funcional y robusta del nuevo algoritmo.

Tras esto pasé a diseñar lo que fue la versión definitiva del algoritmo. Tal y como explicaremos posteriormente, esta versión hacia uso de un pequeño algoritmo auxiliar para localizar la posición relativa de un punto con respecto a un símplice, además de usando un pre-procesamiento con información topológica, fue posible crear un algoritmo funcional y robusto.

Partiendo de este algoritmo realicé varias implementaciones, tales como:

- Inclusión de un solo punto usando sumas atómicas.
- Inclusión de un solo punto usando sumas por reducción.
- Inclusión de un conjunto de puntos.
- Inclusión de un conjunto de puntos con paralelismo dinámico.

Aun habiendo implementado una versión con paralelismo dinámico, dado que esta característica está especialmente pensada para algoritmos recursivos, los resultados en tiempo no fueron muy buenos, es por esto que no se incluye en el proyecto.

Tras la implementación y prueba de las distintas versiones de los algoritmos se pasó a implementar las versiones de comparación. Estas fueron:

- Inclusión de un punto mono-hilo.
- Inclusión de un conjunto de puntos mono-hilo.
- Inclusión de un punto multi-hilo.
- Inclusión de un conjunto de puntos multi-hilo.
- Algoritmo GPGPU de A. Rueda y L. Alvarado (11).

3.3 POSICIÓN RELATIVA DE UN PUNTO EN UN 3-SÍMPLEX

Este algoritmo es una pieza básica del algoritmo diseñado en este proyecto. Básicamente, dado un punto P, determinamos en que elemento del símplex se encuentra.

En la Figura 21 se muestra la nomenclatura escogida para el 3-símplex durante todo el proyecto.

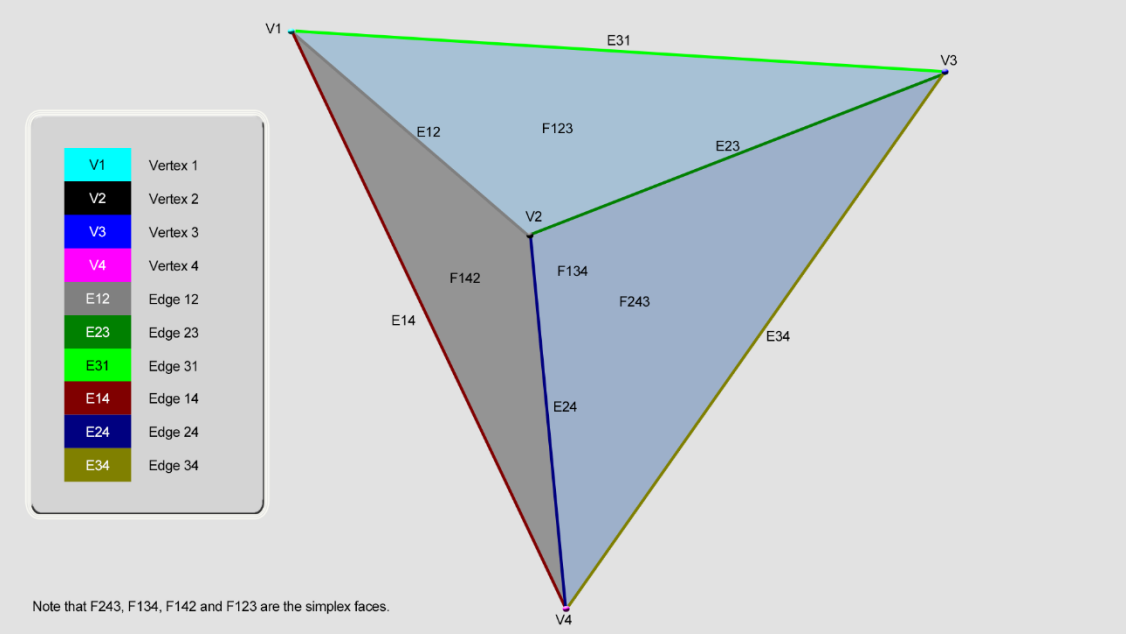

Figura 21. Nomenclatura del 3-símplex

Suponiendo que $(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$ son las coordenadas, se define en la Ecuación 7 una función cuyo valor numérico indicará la casuística en que nos encontramos. Cada valor numérico corresponde a un elemento del 3-simplex, y su correspondencia se plasma en la Tabla 3.

$$f(\lambda_0, \lambda_1, \lambda_2, \lambda_3) = \begin{cases} 0\ si\ \lambda_0 < 0 \vee \lambda_1 < 0 \vee \lambda_2 < 0 \vee \lambda_3 < 0 \\ 1\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 2\ si\ \lambda_0 = 1 \wedge \lambda_1 = 0 \wedge \lambda_2 = 0 \wedge \lambda_3 = 0 \\ 3\ si\ \lambda_0 = 0 \wedge \lambda_1 = 1 \wedge \lambda_2 = 0 \wedge \lambda_3 = 0 \\ 4\ si\ \lambda_0 = 0 \wedge \lambda_1 = 0 \wedge \lambda_2 = 1 \wedge \lambda_3 = 0 \\ 5\ si\ \lambda_0 = 0 \wedge \lambda_1 = 0 \wedge \lambda_2 = 0 \wedge \lambda_3 = 1 \\ 6\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 = 0 \wedge \lambda_3 = 0 \\ 7\ si\ \lambda_0 = 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 = 0 \\ 8\ si\ \lambda_0 > 0 \wedge \lambda_1 = 0 \wedge \lambda_2 > 0 \wedge \lambda_3 = 0 \\ 9\ si\ \lambda_0 > 0 \wedge \lambda_1 = 0 \wedge \lambda_2 = 0 \wedge \lambda_3 > 0 \\ 10\ si\ \lambda_0 = 0 \wedge \lambda_1 > 0 \wedge \lambda_2 = 0 \wedge \lambda_3 > 0 \\ 11\ si\ \lambda_0 = 0 \wedge \lambda_1 = 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 12\ si\ \lambda_0 = 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 13\ si\ \lambda_0 > 0 \wedge \lambda_1 = 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 14\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 = 0 \wedge \lambda_3 > 0 \\ 15\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 = 0 \end{cases}$$

Ecuación 7. $f(\lambda)$ de un punto en un 3-simplex

$f(\lambda)$	$Posición$
0	$Fuera$
1	$Dentro$
2	V_1
3	V_2
4	V_3
5	V_4
6	E_{12}
7	E_{23}
8	E_{31}
9	E_{14}
10	E_{24}
11	E_{34}
12	F_{234}
13	F_{134}
14	F_{124}
15	F_{123}

Tabla 3. Posición relativa del punto según $f(\lambda)$

Dado que para nuestro algoritmo solo queremos diferenciar los casos en el que el punto está fuera, cuando está dentro y cuando está en la frontera, no es necesario una función que diferencie entre tantas casuísticas, es por eso que en la Ecuación 8 se describe la función que finalmente se utiliza para la construcción del algoritmo.

Del mismo modo en la Tabla 4 se describe el significado de cada valor numérico.

$$g(\lambda_0,\lambda_1,\lambda_2,\lambda_3) = \begin{cases} 0\ si\ \lambda_0 < 0 \vee \lambda_1 < 0 \vee \lambda_2 < 0 \vee \lambda_3 < 0 \\ 1\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 5\ si\ \lambda_0 = 1 \wedge \lambda_1 = 0 \wedge \lambda_2 = 0 \wedge \lambda_3 = 0 \\ 5\ si\ \lambda_0 = 0 \wedge \lambda_1 = 1 \wedge \lambda_2 = 0 \wedge \lambda_3 = 0 \\ 5\ si\ \lambda_0 = 0 \wedge \lambda_1 = 0 \wedge \lambda_2 = 1 \wedge \lambda_3 = 0 \\ 0\ si\ \lambda_0 = 0 \wedge \lambda_1 = 0 \wedge \lambda_2 = 0 \wedge \lambda_3 = 1 \\ 5\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 = 0 \wedge \lambda_3 = 0 \\ 5\ si\ \lambda_0 = 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 = 0 \\ 5\ si\ \lambda_0 > 0 \wedge \lambda_1 = 0 \wedge \lambda_2 > 0 \wedge \lambda_3 = 0 \\ 2\ si\ \lambda_0 > 0 \wedge \lambda_1 = 0 \wedge \lambda_2 = 0 \wedge \lambda_3 > 0 \\ 3\ si\ \lambda_0 = 0 \wedge \lambda_1 > 0 \wedge \lambda_2 = 0 \wedge \lambda_3 > 0 \\ 4\ si\ \lambda_0 = 0 \wedge \lambda_1 = 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 6\ si\ \lambda_0 = 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 6\ si\ \lambda_0 > 0 \wedge \lambda_1 = 0 \wedge \lambda_2 > 0 \wedge \lambda_3 > 0 \\ 6\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 = 0 \wedge \lambda_3 > 0 \\ 5\ si\ \lambda_0 > 0 \wedge \lambda_1 > 0 \wedge \lambda_2 > 0 \wedge \lambda_3 = 0 \end{cases}$$

Ecuación 8. $g(\lambda)$ de un punto en un 3-simplex

$f(\lambda)$	$Posición$
0	$Fuera$
1	$Dentro$
2	E_{14}
3	E_{24}
4	E_{34}
5	$Frontera$
6	$F_{234}, F_{134}, F_{124}$

Tabla 4. Posición relativa del punto según $g(\lambda)$

Es importante notar que en el caso en que el punto P sea igual al vértice V_4, el punto se considera directamente fuera. Pues para evitar tener que manejar casuísticas adicionales este vértice será el común de la descomposición simplicial y se escogerá fuera del AABB del poliedro.

3.4 DESCRIPCIÓN DEL ALGORITMO

Como ya hemos explicado anteriormente el algoritmo desarrollado hace uso de un algoritmo auxiliar descrito en la sección anterior. Aun así, no es esta la parte fundamental del algoritmo.

La parte fundamental del algoritmo que permite eliminar la divergencia del mismo es la generación de una máscara binaria que encapsula todas las posibles casuísticas de interés de un punto en coordenadas baricéntricas.

Siendo $\lambda_i \ \forall i \ \in [0,3]$ una componente de las coordenadas baricéntricas del punto P, al compararla es posible realizar 3 comparaciones $\lambda_i > 0$, $\lambda_i < 0$ y $\lambda_i = 0$. Dichas comparaciones son las necesarias para evaluar la función $\mathrm{g}(\lambda)$, con la cual se puede obtener la posición relativa del punto P.

Si definimos las funciones $A(\lambda_i)$, $B(\lambda_i)$, tal y como se muestran en la Ecuación 9, es posible encapsular las casuísticas de λ_i en dos variables booleanas.

$$A(\lambda_i) = \begin{cases} 1 \ si \ \lambda_i > 0 \\ 0 \ en \ otro \ caso \end{cases} \qquad B(\lambda_i) = \begin{cases} 1 \ si \ \lambda_i = 0 \\ 0 \ en \ otro \ caso \end{cases}$$

Ecuación 9. Funciones $A(\lambda_i)$ y $B(\lambda_i)$

Por tanto la máscara binaria generada estará formada por tantos de estos pares de variables booleanas como componentes tengan las coordenadas baricéntricas. De este modo la máscara binaria representará todas las casuísticas de $\mathrm{g}(\lambda)$.

Definimos por tanto para el caso que nos ocupa la función generadora de la máscara binaria $\mathrm{m}(\lambda)$ tal y como se muestra en la Ecuación 10.

$$\begin{aligned} & m(\lambda_0, \lambda_1, \lambda_2, \lambda_3) = \\ & = A(\lambda_0) \cdot 2^7 + B(\lambda_0) \cdot 2^6 \\ & + A(\lambda_1) \cdot 2^5 + B(\lambda_1) \cdot 2^4 \\ & + A(\lambda_2) \cdot 2^3 + B(\lambda_2) \cdot 2^2 \\ & + A(\lambda_0) \cdot 2^1 + B(\lambda_0) \cdot 2^0 \end{aligned}$$

Ecuación 10. Función de máscara binaria $m(\lambda)$

Una de las incompatibilidades para ser no-divergente del algoritmo descrito en el artículo de F. Feito y J.C. Torres consiste en que para identificar las casuísticas especiales de punto en arista, vértice o cara del triángulo original necesita de bloques condicionales así como de estructuras de almacenamiento dinámico.

Para solventar esto, en el algoritmo desarrollado se hace uso de un formato pre-procesado de la B-Rep original.

Esta nueva representación, ITR, contiene un vector de nodos correspondientes a la descomposición simplicial de la B-Rep. Cada uno de estos nodos contiene la matriz de conversión a coordenadas baricéntricas, así como 7 factores los cuales son la clave para la ejecución no-divergente del algoritmo.

Estos factores definen las componentes de la sumatoria que determinarán el resultado de la ejecución del algoritmo, de tal modo que dependiendo de la posición relativa del punto evaluado con el 3-simplex el componente de la sumatoria variará.

La elección de uno de estos 7 factores viene determinada por la función $g(\lambda)$ mostrada en la Ecuación 8, siendo descrito el significado de los 7 factores en la Tabla 4.

Aun así, la aplicación del a función $g(\lambda)$ por sí sola no nos garantiza la no-divergencia del algoritmo, pues una función definida a intervalos requiere el uso de condicionales para su evaluación.

Aquí es donde la función de máscara binaria $m(\lambda)$ definida en la Ecuación 10, entra a formar parte del algoritmo. Dado que dicha función cubre las casuísticas mayor, menor o igual para cada componente de las coordenadas baricéntricas, es posible por tanto identificar únicamente la posición relativa del punto evaluado con respecto a un 3-simplex.

Aunque se podría aplicar directamente la función $m(\lambda)$ y obtener un factor sumatorio de entre los 256 posibles de $m(\lambda)$ directamente desde el nodo, dado que muchos de los valores se repiten y otros tantos valores resultantes de $m(\lambda)$ son indeterminaciones, durante la fase de desarrollo del algoritmo decidí introducir un tercer elemento, una tabla de asignaciones de casuísticas.

Esta tabla o función constante $h(k)$, es la que nos permite relacionar las funciones $m(\lambda)$ y $g(\lambda)$ de tal modo que para cada valor de $m(\lambda)$ en la tabla hay un valor de $g(\lambda)$.

Con el objetivo de dar una definición formal del algoritmo se define la función $v(k)$, la cual relaciona los factores con las casuísticas de $g(\lambda)$.

En el siguiente apartado se describirá la forma de calcular el valor de cada factor, y por consiguiente, la función $v(k)$.

Una vez realizadas estas definiciones previas es posible dar una definición formal del algoritmo, de tal modo que siendo

$P = (x, y, z), el\ punto\ a\ evaluar.$

$n, el\ número\ de\ nodos\ de\ la\ representación\ ITR.$

$B = \{B_i \mid \forall i \in \{1 \dots n\}\}, las\ matrices\ de\ transformación\ de\ los\ nodos.$

$V = \{v_i(k) \mid \forall i \in \{1 \dots n\}\}, el\ conjunto\ funciones\ v(k)\ para\ cada\ nodo.$

Definimos la función de evaluación del algoritmo $e(P,\ B, V, n)$ como

$$e(P, B, V, n) = \sum_{i=1}^{n} v_i(h(m(B_i \times P)))$$

Si $e(P, B, V, n) = 1$, el punto se encuentra dentro de la B-Rep.

Si $e(P, B, V, n) = \infty$, el punto se encuentra en la frontera de la B-Rep.

En otro caso, el punto está fuera de la B-Rep.

A continuación, en la Figura 22 se puede ver un pseudocódigo de la implementación utilizada durante las pruebas del TFG.

```
m_11 = pNode->Transform.m_11
m_12 = pNode->Transform.m_12
m_13 = pNode->Transform.m_13
m_21 = pNode->Transform.m_21
m_22 = pNode->Transform.m_22
m_23 = pNode->Transform.m_23
m_31 = pNode->Transform.m_31
m_32 = pNode->Transform.m_32
m_33 = pNode->Transform.m_33

s = m_11*_x + m_12*_y + m_13*_z
t = m_21*_x + m_22*_y + m_23*_z
u = m_31* x + m_32*_y + m_33*_z
v = ONE_VALUE - s
v-=(t+u)

_mask = (FPUEX_GTCOMPARE(s) << 7) | (FPUEX_ZCOMPARE(s) << 6)
      | (FPUEX_GTCOMPARE(t) << 5) | (FPUEX_ZCOMPARE(t) << 4)
      | (FPUEX_GTCOMPARE(u) << 3) | (FPUEX_ZCOMPARE(u) << 2)
      | (FPUEX_GTCOMPARE(v) << 1) | FPUEX_ZCOMPARE(v)

_val+=pNode->ItemFactors[(*gd_ConstantData).MaskTable[_mask]]
```

Figura 22. Pseudocódigo del algoritmo

3.5 CÁLCULO DE LOS FACTORES DEL ALGORITMO

Como ya hemos explicado en la anterior sección, los factores de los nodos del algoritmo tienen un papel fundamental para la evaluación de la función del algoritmo, es por eso que en esta sección se el cálculo de los mismo.

Partimos de una representación limpia de un modelo 2-manifold. Hay varias representaciones que pueden ser válida para su conversión a ITR, dos de ellas son la representación de arista alada y la representación B-Rep.

En última instancia los datos más importantes para la generación de la representación ITR son los datos topológicos del modelo, es por esto que al utilizar una representación de arista alada los cálculos se facilitan bastante.

Aun facilitando más el cómputo la representación de arista alada en este proyecto he decidido partir de representaciones B-Rep, pues es el tipo de representación al que está enfocado el proyecto, facilitan la posterior depuración en las pruebas realizadas, además de que son las representaciones más comunes para renderizado actualmente.

Una vez tenemos la información topológica, así como geométrica del modelo podemos proceder al cálculo de los factores de cada nodo.

En primer lugar es imperante realizar el cálculo de los signos de los 3-simplex de la descomposición simplicial del modelo, tal y como se muestra en la Ecuación 2.

Posteriormente se generan dos vectores, $VPos$ y $VNeg$, los cuales contienen respectivamente el número de caras positivas y negativas adyacentes a cada vértice del modelo.

Tras esto se genera un vector, $VFactors$, el cual se calcula como se muestra en el siguiente pseudocódigo.

```
for i = 0; i < Número de vértices; ++i
       _pos = VPos[i] > 0;
       _neg = VNeg[i] > 0;
       if !(_pos ^ _neg)
              VFactors[i] = 0;
       else if _pos
              VFactors[i] = 1.0 / VPos[i];
       else if _neg
              VFactors[i] = -1.0 / VNeg[i];
```

Por último lugar, se define realmente el valor de los factores de los nodos del siguiente modo

$$v(k) = \begin{cases} 0, & k = 0 \\ sign, & k = 1 \\ VFactors[F_{v0}], & k = 2 \\ VFactors[F_{v1}], & k = 3 \\ VFactors[F_{v2}], & k = 4 \\ \infty, & k = 5 \\ \dfrac{sign}{2}, & k = 6 \end{cases}$$

Siendo:

$sign$, el signo del nodo.

F_{v0}, F_{v1} y F_{v3}, los índices de los vértices de la cara generadora del nodo asociado al 3-simplex.

3.6 IMPLEMENTACIONES

El pseudocódigo mostrado en el anterior apartado corresponde al de una de las implementaciones realizadas en el algoritmo. Dicha implementación solo evalúa un único punto contra una representación B-Rep, lo cual como veremos en el apartado de resultados no supone una mejora importante sobre la ejecución de CPU.

Por lo anteriormente comentado tomé la decisión de realizar varias implementaciones con el objetivo de mejorar el rendimiento.

3.6.1 Implementación atómica contra un solo punto

Esta implementación es la que hemos descrito con anterioridad. Cada hilo del grid calcula el factor de cada nodo y añade su valor al resultado del algoritmo.

Uno de los principales problemas que presenta este enfoque es que, al ejecutarse varios hilos de forma paralela, es una necesidad que la operación de añadir el valor del factor al resultado del algoritmo sea atómica. De lo contrario, cuando varios hilos intenten escribir el resultado, este tiene una alta probabilidad de no ser consistente.

Dado que en CUDA todos los hilos de un WARP ejecutan la misma instrucción, cuando se realiza la suma atómica lo que ocurre es que estas han de realizarse de manera secuencial, en cadenas del tamaño del WARP (32 por defecto en las arquitecturas utilizadas), lo que supone un bloqueo en la ejecución.

En el siguiente apartado se describirá una mejora que permita evitar las sumas atómicas.

3.6.2 Implementación de reducción contra un solo punto

En el anterior apartado ya se ha explicado la problemática que presenta el uso de operaciones atómicas, por esto desarrollé una versión la cual no utilizaba sumas atómicas.

Esta versión usa un método bastante común en paralelización masiva conocido como suma por reducción.

Este método se basa en el hecho de que el acceso a la memoria compartida de CUDA es mucho menor que el tiempo de acceso a la memoria global.

Consiste en aplicar un algoritmo divide y vencerás, de tal modo que únicamente se producen $\log_2 n$ bloqueos.

Existen varios métodos para aplicar la suma por reducción. En esta implementación el enfoque seguido es el de direccionamiento secuencial, el cual se describe gráficamente en la Figura 23.

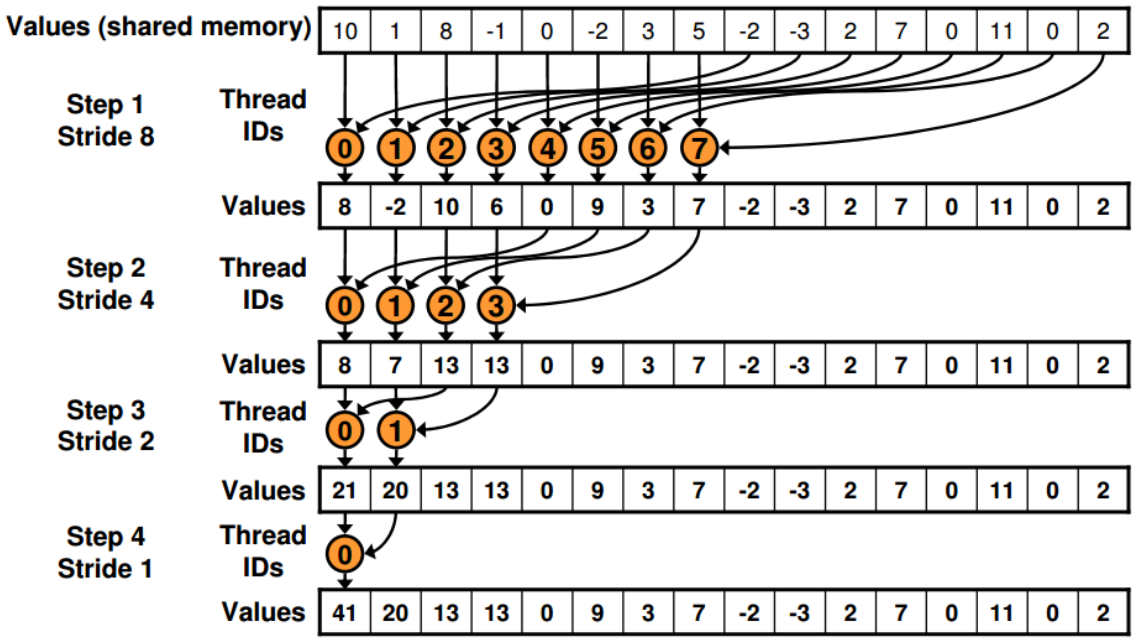

Figura 23. Direccionamiento secuencial

3.6.3 Implementación para un conjunto de puntos

Las dos implementaciones descritas anteriormente se centran en evaluar un único punto, sin embargo en muchos de los problemas en los que se requiere de este tipo de algoritmo se evalúan múltiples puntos al mismo tiempo.

Es por esto que he implementado una versión que evalúa múltiples puntos sobre un mismo modelo B-Rep. En la Figura 24 se muestra el pseudocódigo de la implementación.

```
for(i = (*gd_ConstantData).NodeCount; i >= 0; --i)
  Node *pNode = &pNodes[i]
  m_11 = pNode->Transform.m_11
  m_12 = pNode->Transform.m_12
  m_13 = pNode->Transform.m_13
  m_21 = pNode->Transform.m_21
  m_22 = pNode->Transform.m_22
  m_23 = pNode->Transform.m_23
  m_31 = pNode->Transform.m_31
  m_32 = pNode->Transform.m_32
  m_33 = pNode->Transform.m_33

  s = m_11*_x + m_12*_y + m_13*_z
  t = m_21*_x + m_22*_y + m_23*_z
  u = m_31*_x + m_32*_y + m_33*_z
  v = ONE_VALUE - s
  v-=(t+u)

  _mask = (FPUEX_GTCOMPARE(s) << 7) | (FPUEX_ZCOMPARE(s) << 6)
        | (FPUEX_GTCOMPARE(t) << 5) | (FPUEX_ZCOMPARE(t) << 4)
        | (FPUEX_GTCOMPARE(u) << 3) | (FPUEX_ZCOMPARE(u) << 2)
        | (FPUEX_GTCOMPARE(v) << 1) | FPUEX_ZCOMPARE(v)

  _val+=pNode->ItemFactors[(*gd_ConstantData).MaskTable[_mask]]
```

Figura 24. Pseudocódigo de la implementación para un conjunto de puntos.

Como se puede ver, en este caso no existe el problema de conflicto de escritura a la memoria global, pues cada hilo escribe un valor distinto al finalizar la ejecución.

Tal y como se verán en los resultados de las ejecuciones para esta implementación la ganancia es altamente superior a las anteriores implementaciones, siendo mayor la ganancia cuanto mayor sea la muestra de puntos evaluados.

3.6.4 Implementación para la voxelización CSG

Con el objetivo de demostrar una aplicación práctica del algoritmo diseñado, he creado una implementación sobre CUDA para obtener la voxelización de un modelo CSG, cuyas primitivas son representación B-Rep.

Dicha implementación es bastante similar a la anterior, ya que el único trabajo que realiza la GPU es el de evaluar un conjunto de puntos divididos de forma homogénea en un sub-espacio de $\mathbb{R}^3$.

Aun así, la clave para lograr una alta eficiencia en esta implementación no reside en este caso en la parte implementada mediante GPGPU, sino la forma de enfocar la evaluar la evaluación del árbol CSG.

La anterior mencionada eficiencia es debido a que a partir de los resultados de inclusión para cada primitiva del CSG, se genera una máscara binaria en el cual un valor nulo significa que un punto concreto no corresponde a la primitiva evaluada en ese caso.

Tras generar todas las máscaras binarias, obtener el conjunto de vóxeles resultantes se reduce simplemente a la aplicación operaciones lógicas en cada nodo del árbol CSG evaluado, tras como se muestra en pseudocódigo de la Figura 25.

```
StringPointer _ptr(m_NprString)
while (StringPointer _token = _ptr.Get(' '))
  if (_token.IsUInt())
    m_Stack.push(m_Primitives[_token.ToUInt()])
  else
    Bitmask _op2 = m_Stack.top()  m_Stack.pop()
    Bitmask _op1 = m_Stack.top()  m_Stack.pop()

    switch (CsgOperator(_token.ToChar()))
      case UNION_OPERATOR: m_Stack.push(_op1.Or(_op2))  break
      case INTERSECTION_OPERATOR: m_Stack.push(_op1.And(_op2))  break
      case DIFFERENCE_OPERATOR: m_Stack.push(_op1.Diff(_op2))  break
```

Figura 25. Pseudocódigo de la evaluación del árbol CSG.

Se muestra en la Figura 26 un ejemplo de modelo CSG resultado de la implementación presentada.

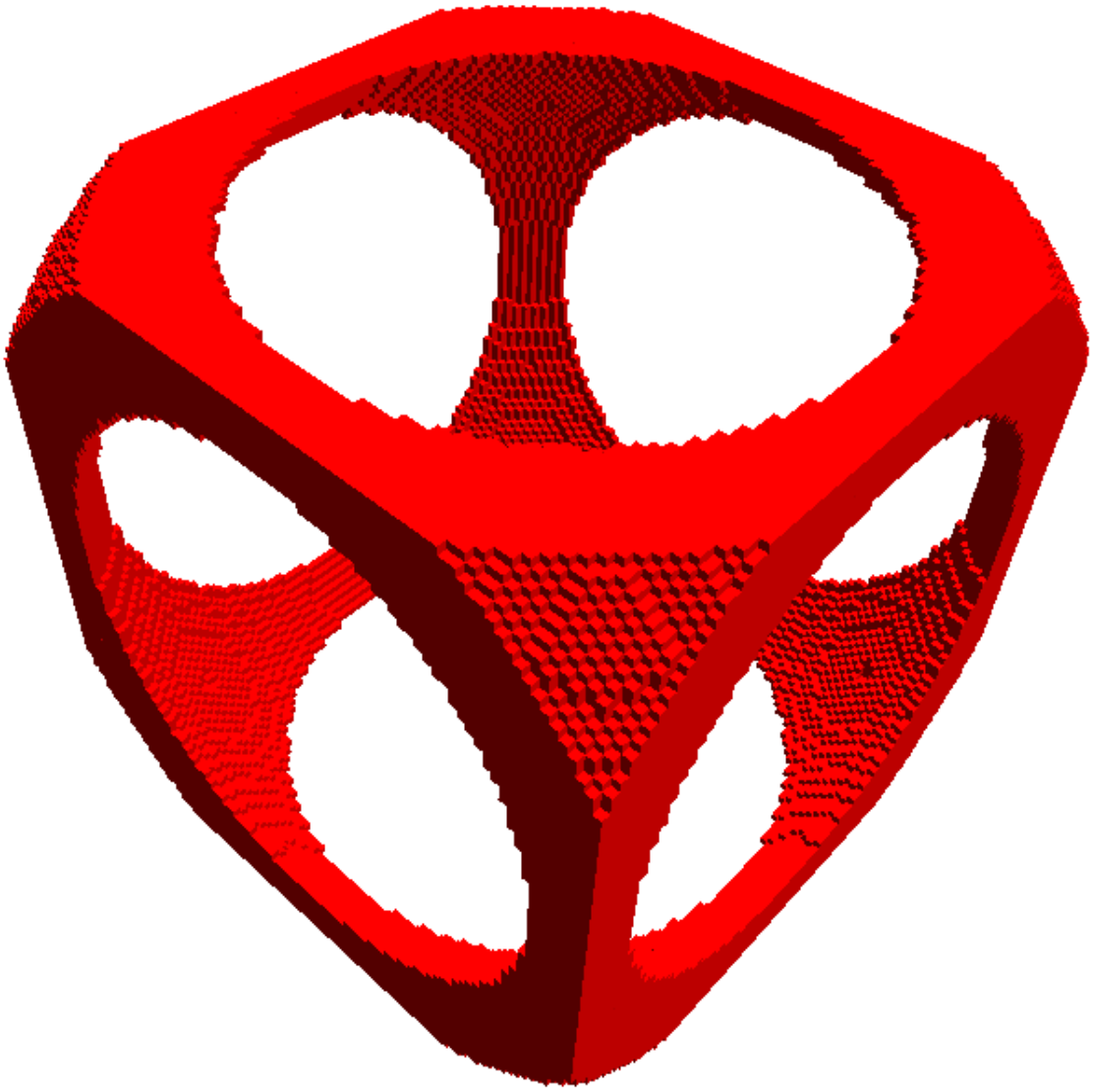

Figura 26. Modelo CSG voxelizado mediante la descrita implementación.

3.6.5 Otras implementaciones

Con el objetivo de realizar comparaciones con las distintas implementaciones GPGPU, he realizado otras implementaciones sobre CPU, así como GPU, las cuales describo a continuación.

En primer lugar implementé una versión monohilo del algoritmo presentado sobre CPU. Dicha implementación evalúa múltiples puntos en un sobre un solo hilo de CPU.

Es importante notar que la anterior implementación a su vez proporciona datos estadísticos suficientes como para que no sea necesario realizar una implementación monohilo de evaluación de un punto contra un poliedro.

La siguiente implementación es una adaptación del algoritmo de evaluación de un solo punto contra un poliedro en CPU, utilizando múltiples hilos.

Otra implementación realizada muy similar a la anterior que se utiliza para comparar es la adaptación del algoritmo de evaluación de un conjunto de puntos contra un poliedro en CPU, utilizando múltiples hilos.

Para finalizar, se ha utilizado una implementación GPGPU de A. Rueda y L. Alvarado para un conjunto de puntos, con el objetivo de comparar el algoritmo presentado con otra implementación similar.

4 RESULTADOS

4.1 INTRODUCCIÓN

Una parte fundamental en los proyectos de carácter teórico experimental es la de la obtención de datos estadísticos para inferir una conclusión.

En el caso de este proyecto, es por tanto un aspecto fundamental, el de ejecutar las distintas implementaciones realizadas del algoritmo sobre distintos conjuntos de datos.

Más aún será necesario ejecutar los mismos conjuntos de datos sobre las implementaciones de otros algoritmos que realicen la misma funcionalidad que el aquí presentado, con el objetivo final de realizar una comparación en término de recursos computacionales.

Es un hecho obvio la necesidad de comprar los resultados obtenidos de ejecuciones del algoritmo presentado, tanto de algoritmos establecidos, pues el proyecto que tratamos es principalmente de optimización.

4.2 CONSIDERACIONES PREVIAS

En este apartado se describen todos los detalles necesarios a conocer para poder replicar correctamente las pruebas realizadas para la obtención de los datos estadísticos.

4.2.1 Condiciones de ejecución

Durante la ejecución de todas las pruebas realizadas a lo largo del desarrollo del proyecto se ha usado el mismo hardware, así como sistema operativo y configuración del mismo.

Además se ha tratado en la medida de lo posible que el establecer al mínimo el uso de los recursos de la máquina de ejecución antes de llevar a cabo las pruebas.

A continuación se describe el hardware utilizado:

- CPU. Intel® Core™ i5-2500K.
- RAM. 2 x Kingston® KVR13N9S8/4 CL9 4GB
- GPU. ASUS® GTX 660Ti DirectCU.

Cabe notar que las frecuencias de reloj de las anteriores piezas de hardware descritas se han mantenido tal cual venían de manufactura para la realización de las ejecuciones.

Se ha utilizado Microsoft® Windows® 7 Professional como sistema operativo para la máquina que ha realizado las ejecuciones.

Todas las pruebas se han realizado usando CUDA SDK 6.5.

4.2.2 Modelos utilizados durante las pruebas

Es absolutamente necesario seleccionar un conjunto de modelos topológicamente correctos, los cuales se utilizarán para la obtención de las estadísticas de las distintas implementaciones.

Para el proyecto actual he escogido 3 modelos significativos de la plataforma TurboSquid (12), mostrados en la Figura 27, la Figura 28 y la Figura 29.

Cabe notar que para la realización de las pruebas, la topografía de los modelos escogida es irrelevante, pero sin embargo la complejidad geométrica de estos no.

Se han escogido estos tres modelos con una complejidad geométrica dispar y ascendente con el objetivo de demostrar que la ganancia en tiempo es mayor para el algoritmo presentado cuanto mayor sea la complejidad del modelo que se trata.

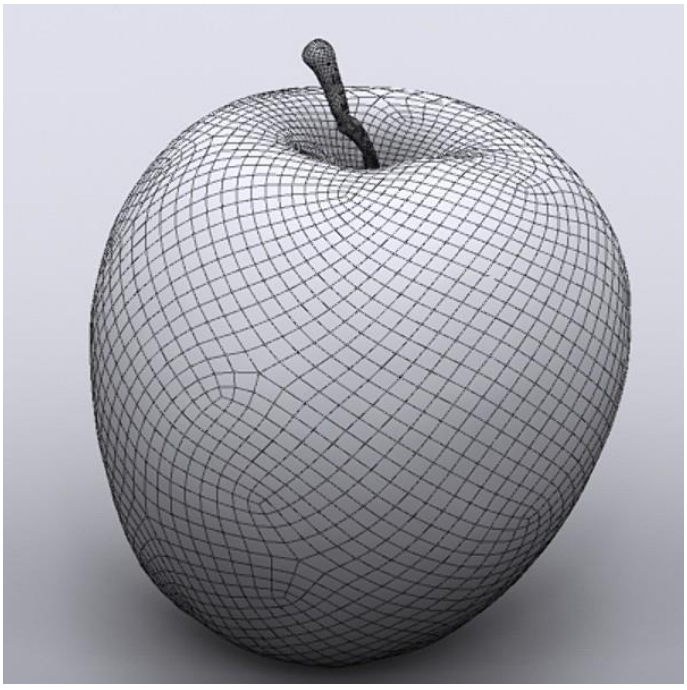

Figura 27. Modelo "Apple" con 13568 triángulos.

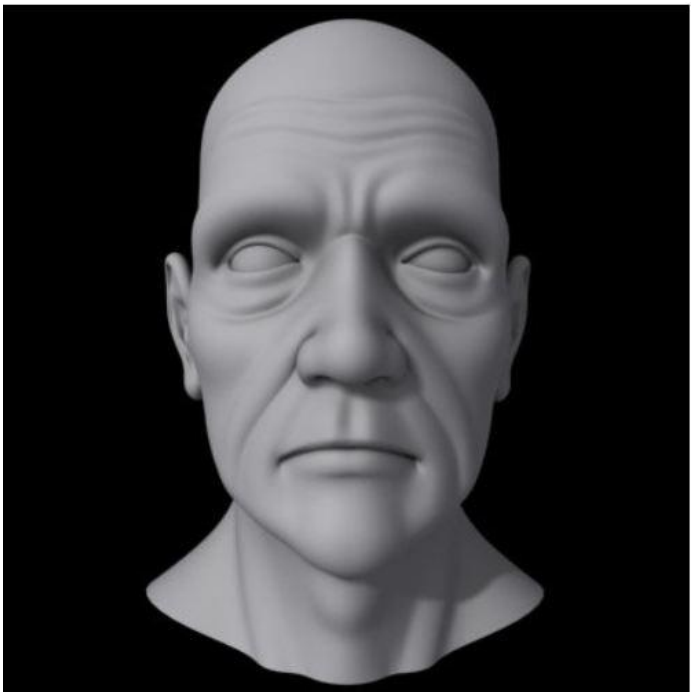

Figura 28. Modelo “Oldman” con 69632 triángulos.

Figura 29. Modelo “Knife” con 133120 triángulos.

4.2.3 Convenciones de ejecución

Se describe a continuación la nomenclatura utilizada para nombrar a las distintas implementaciones, de las cuales se van a obtener datos estadísticos.

- Monohilo para un solo punto: STP
- Monohilo para varios puntos: STG
- Multihilo para un solo punto: MTP
- Multihilo para varios puntos: MTG
- GPGPU para un solo punto: R660
- GPGPU para varios puntos: G660

Además todas las ejecuciones se ejecutan sobre varios conjuntos de puntos distribuido homogéneamente en un sub-espacio de $\mathbb{R}^3$ de 32x32x32, 64x64x64, 128x128x128 y 256x256x256 respectivamente.

4.3 EJECUCIONES DE “STP”

Para poder interpretar los resultados de la ejecución “STP” es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- El “TGlobal” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- El “TIndividual” se refiere a la media de ejecución de tiempo de la implementación por cada punto.

Como se puede observar a lo largo de las distintas ejecuciones, aunque obviamente aumente el tiempo de ejecución global cuanto mayor sea el conjunto de puntos y mayor complejidad geométrica tenga el modelo contra el que se evalúa, el tiempo de ejecución individual tiende a ser inversamente proporcional al tamaño del conjunto, tal y como se puede ver claramente en la Figura 32. Esto es debido a la cache del procesador, puesto que cuanto mayor veces ejecute la evaluación individual, menor probabilidad de fallo tendrá.

En la Tabla 5 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	5,77865E+09	1,75327E+05
64x64x64	4,64098E+10	1,76313E+05
128x128x128	3,70532E+11	1,76654E+05
256x256x256	2,96455E+12	1,76674E+05

Tabla 5. Ejecución “STP” para el modelo “Apple”

En la Figura 30 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Apple”.

Figura 30. Gráfica de la ejecución “STP” para el modelo “Apple”

En la Tabla 6 se muestran los tiempos de ejecución para el modelo “Oldman”.

Resolución	TGlobal	TIndividual
32x32x32	3,03662E+10	9,19581E+05
64x64x64	2,41689E+11	9,21876E+05
128x128x128	1,93423E+12	9,22243E+05
256x256x256	1,55953E+13	9,29422E+05

Tabla 6. Ejecución “STP” para el modelo “Oldman”

En la Figura 31 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Oldman”.

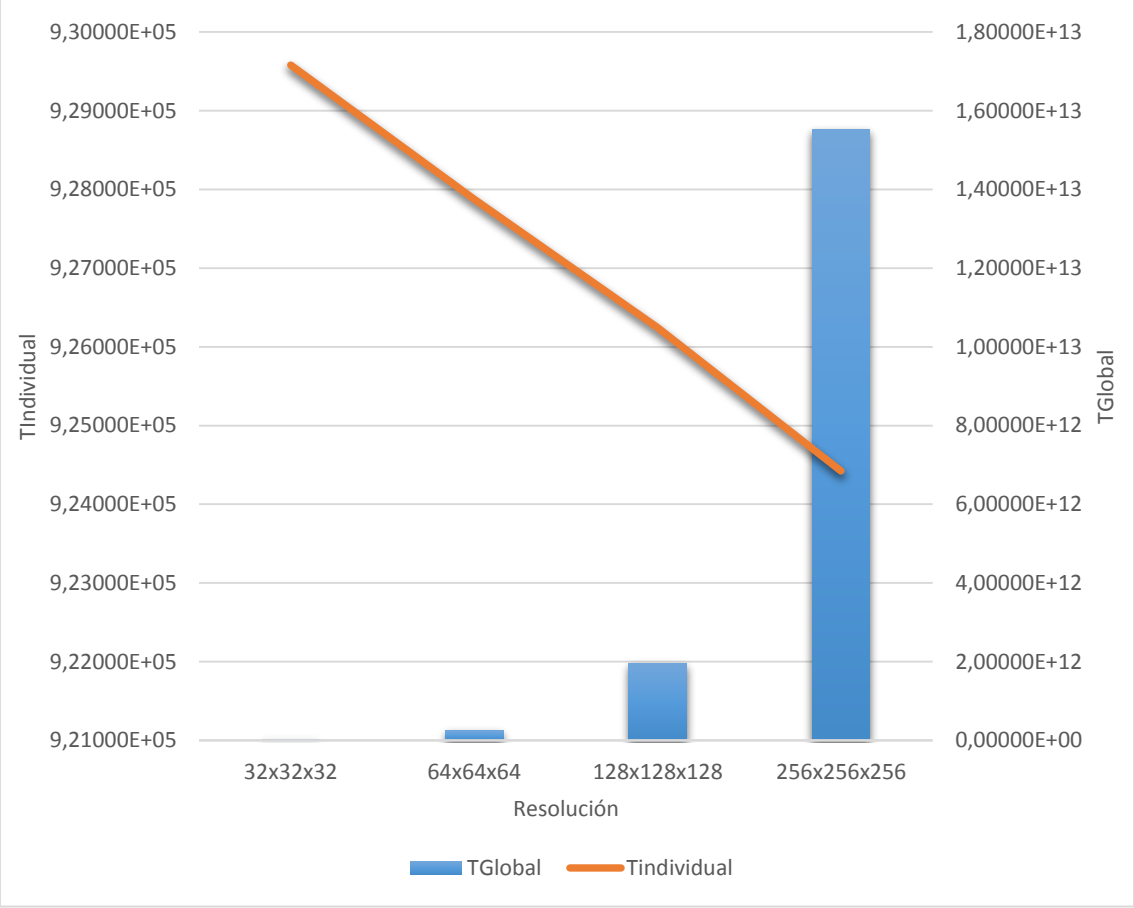

Figura 31. Gráfica de la ejecución “STP” para el modelo “Oldman”

En la Tabla 7 se muestran los tiempos de ejecución para el modelo "Knife".

Resolución	TGlobal	TIndividual
32x32x32	6,94924E+10	2,12043E+06
64x64x64	5,41674E+11	2,06600E+06
128x128x128	4,00519E+12	1,90964E+06
256x256x256	3,19704E+13	1,90589E+06

Tabla 7. Ejecución "STP" para el modelo "Knife"

En la Figura 32 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo "Knife".

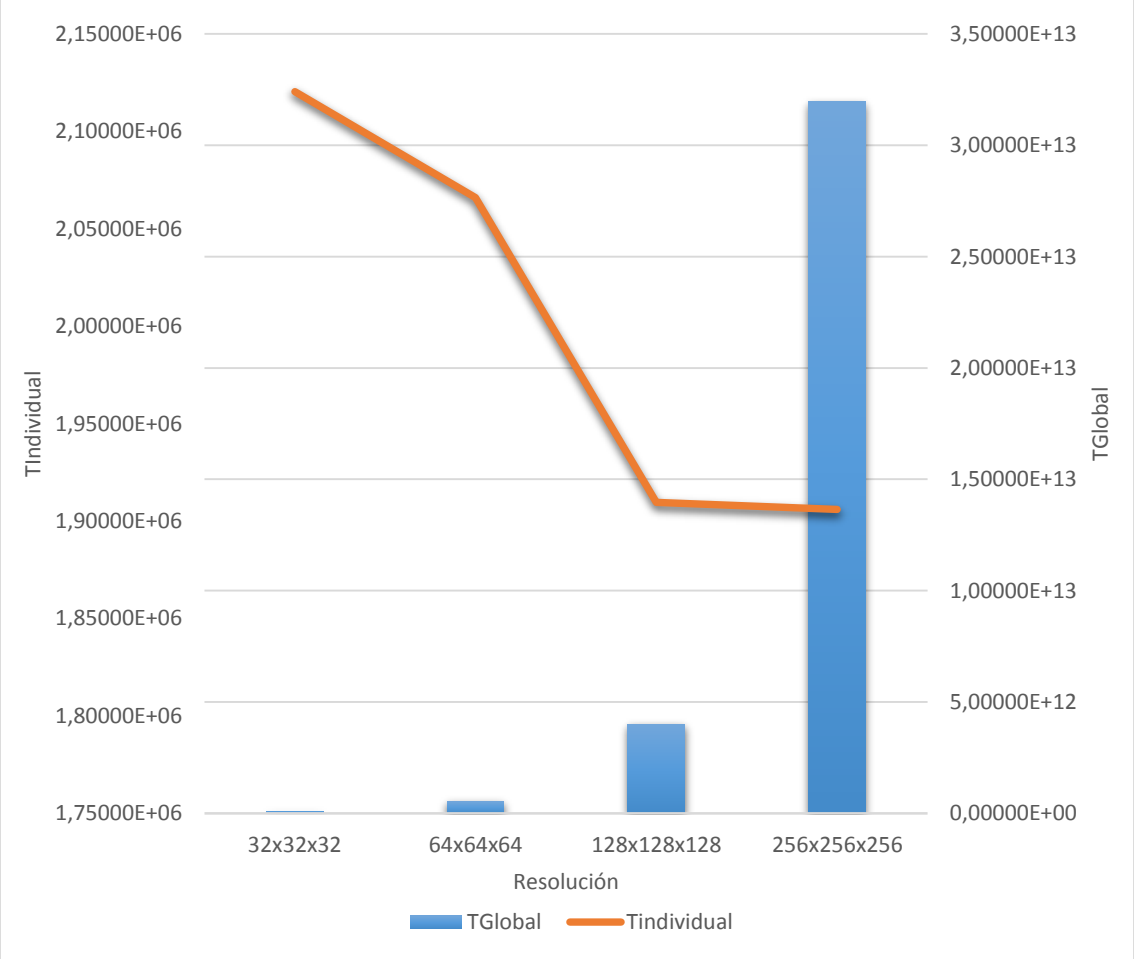

Figura 32. Gráfica de la ejecución "STP" para el modelo "Knife"

4.4 EJECUCIONES DE “STG”

Para poder interpretar los resultados de la ejecución “STG” es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- El “TGlobal” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- El “TIndividual” se refiere a la media de ejecución de tiempo de la implementación por cada punto.

En la Tabla 8 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	5,82841E+09	1,77869E+05
64x64x64	4,65644E+10	1,77629E+05
128x128x128	3,72445E+11	1,77596E+05
256x256x256	2,97523E+12	1,77337E+05

Tabla 8. Ejecución “STG” para el modelo “Apple”

En la Figura 33 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Apple”.

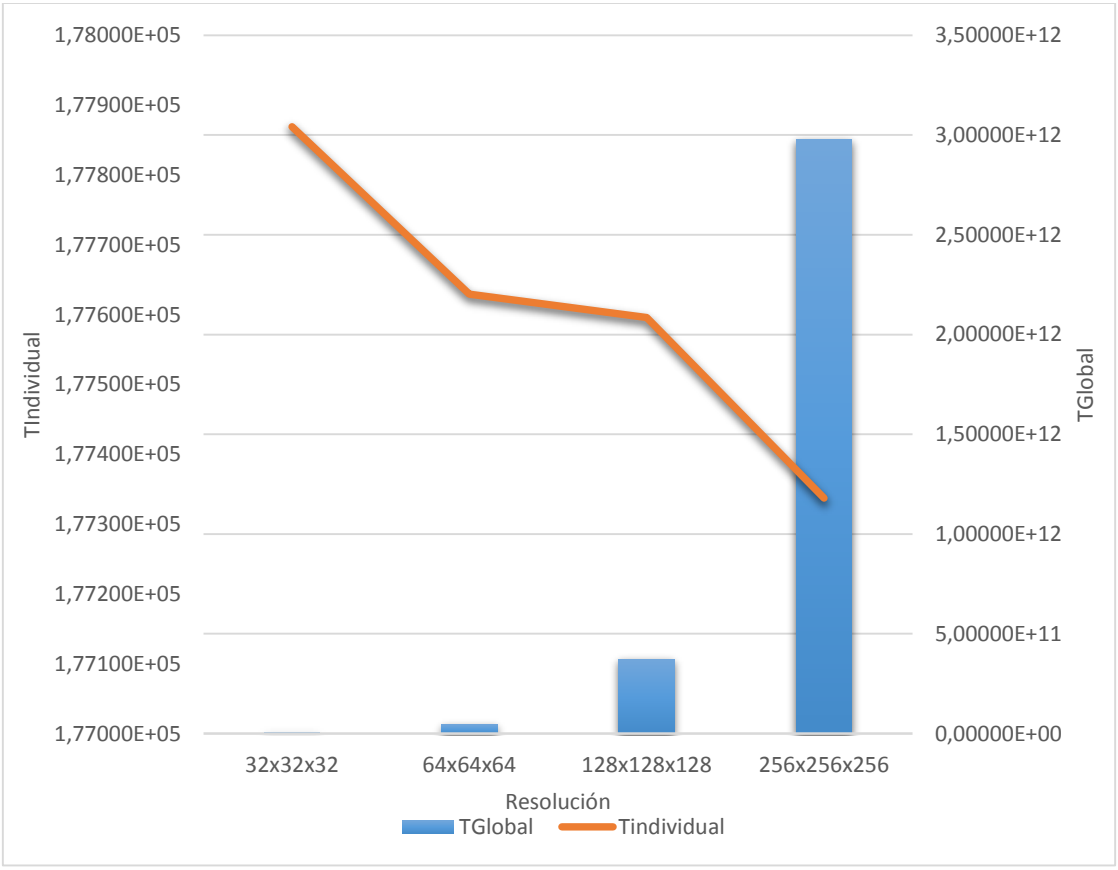

Figura 33. Gráfica de la ejecución “STG” para el modelo “Apple”

En la Tabla 9 se muestran los tiempos de ejecución para el modelo "Oldman".

Resolución	TGlobal	TIndividual
32x32x32	3,03812E+10	9,27162E+05
64x64x64	2,41979E+11	9,26078E+05
128x128x128	1,94588E+12	9,25868E+05
256x256x256	1,55472E+13	9,23686E+05

Tabla 9. Ejecución "STG" para el modelo "Oldman"

En la Figura 34 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo "Oldman".

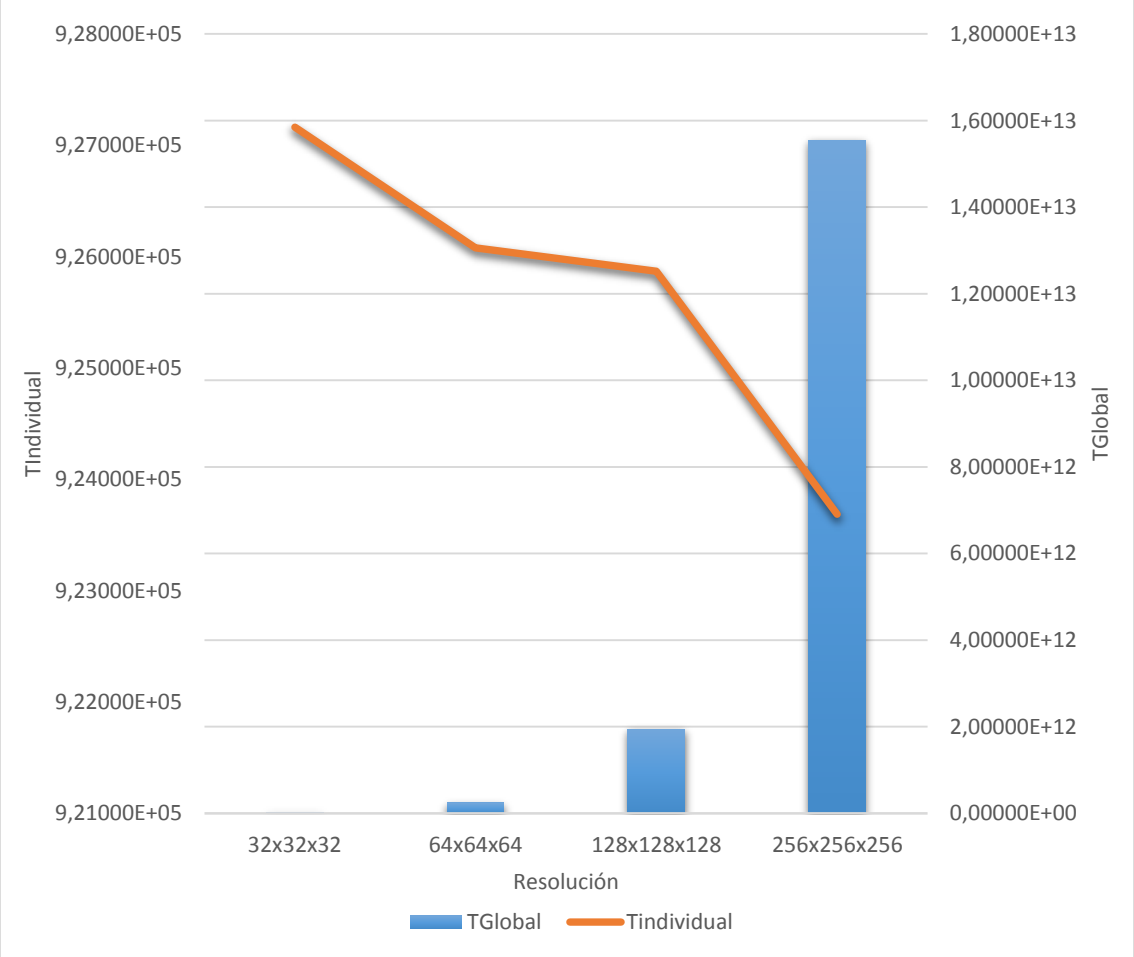

Figura 34. Gráfica de la ejecución "STG" para el modelo "Oldman"

En la Tabla 10 se muestran los tiempos de ejecución para el modelo "Knife".

Resolución	TGlobal	TIndividual
32x32x32	6,84338E+10	2,08843E+06
64x64x64	5,38665E+11	2,05485E+06
128x128x128	4,01574E+12	1,91485E+06
256x256x256	3,19999E+13	1,90735E+06

Tabla 10. Ejecución "STG" para el modelo "Knife"

En la Figura 35 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo "Knife".

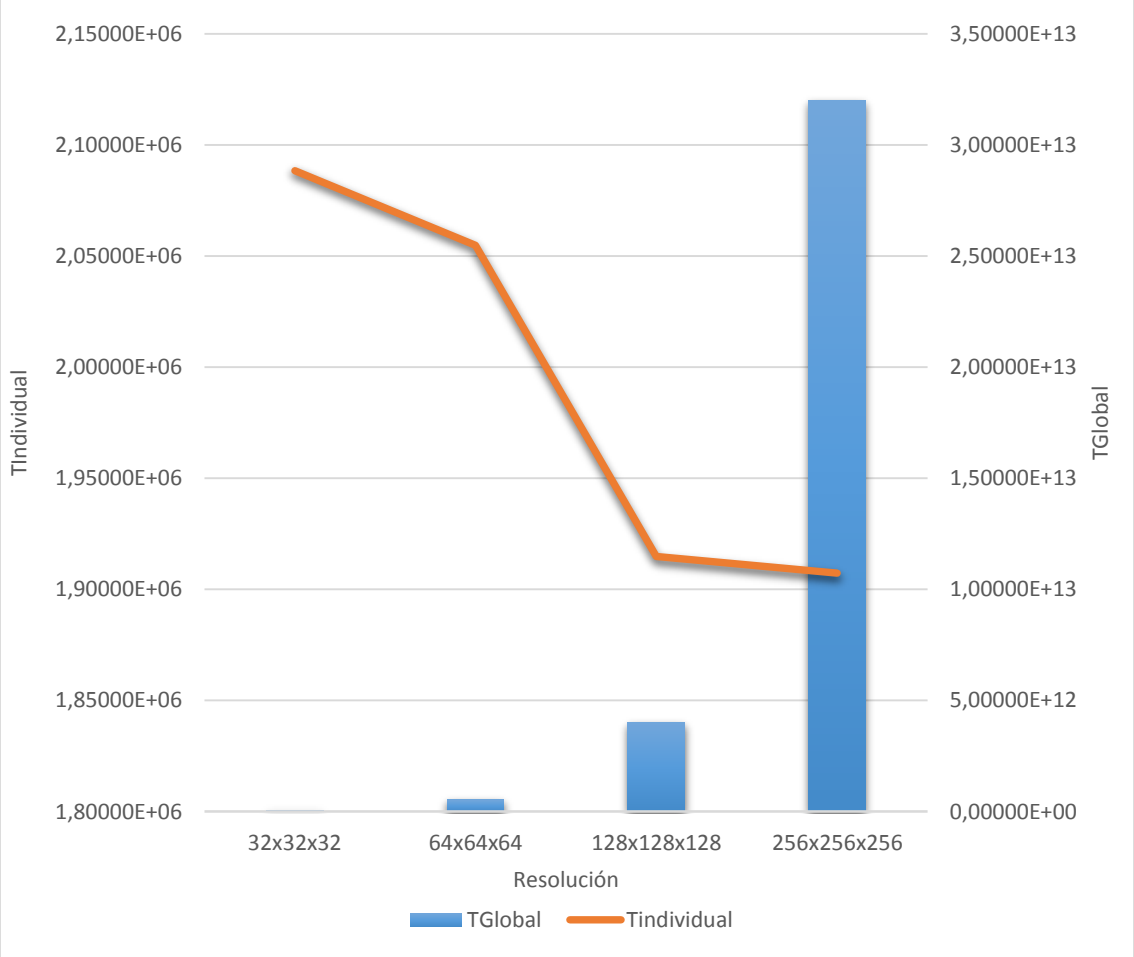

Figura 35. Gráfica de la ejecución "STG" para el modelo "Knife"

4.5 EJECUCIONES DE “MTP”

Para poder interpretar los resultados de la ejecución “MTP” es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- El “TGlobal” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- El “TIndividual” se refiere a la media de ejecución de tiempo de la implementación por cada punto.

Cabe notar que en esta versión se han lanzado 4 hilos simultáneamente, dado que es el número de núcleos del procesador, sabiendo además que este no tiene Hyper-threading.

En la Tabla 11 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	4,77167E+09	6,00547E+04
64x64x64	3,79092E+10	5,94534E+04
128x128x128	3,04425E+11	5,94369E+04
256x256x256	2,50575E+12	5,93462E+04

Tabla 11. Ejecución “MTP” para el modelo “Apple”

En Figura 36 la se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Apple”.

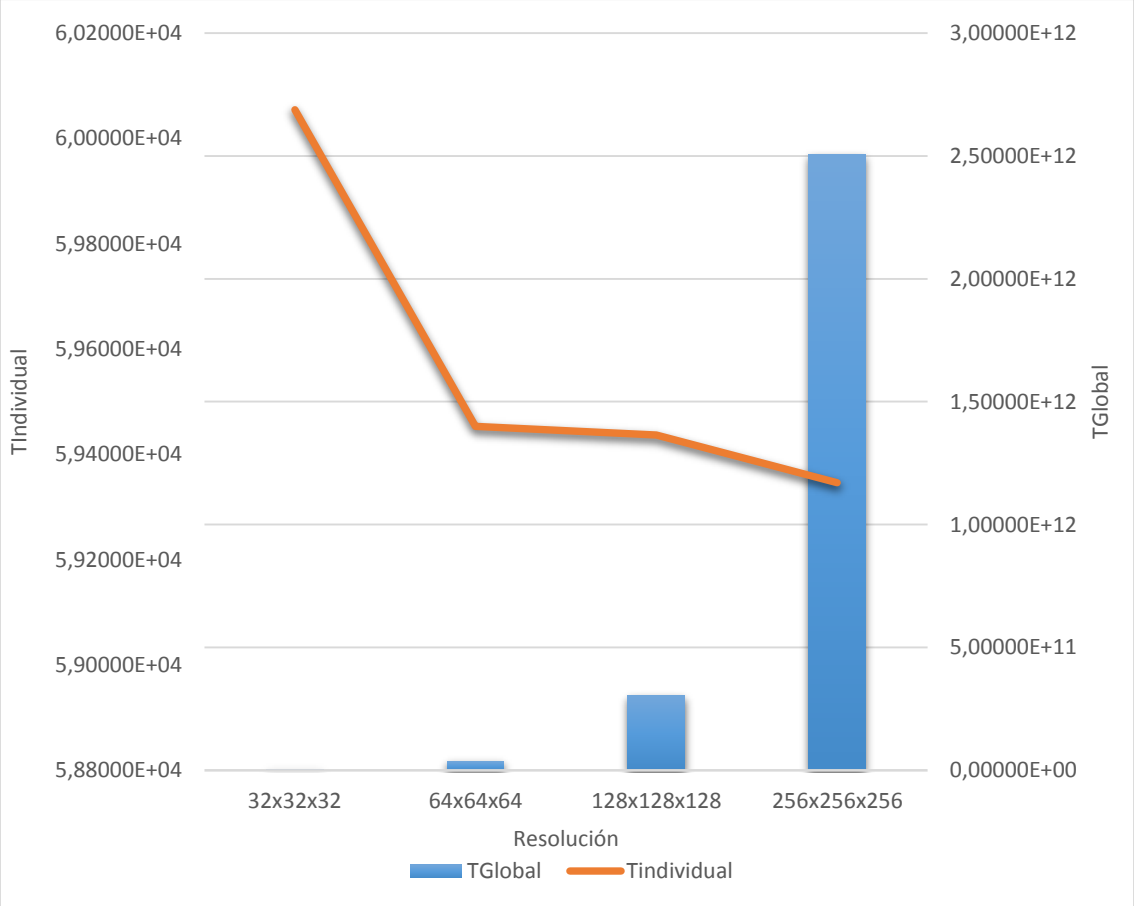

Figura 36. Gráfica de la ejecución “MTP” para el modelo “Apple”

En la se muestran los tiempos de ejecución para el modelo “Oldman”.

Resolución	TGlobal	TIndividual
32x32x32	1,19555E+10	2,79806E+05
64x64x64	9,65888E+10	2,78505E+05
128x128x128	7,65960E+11	2,77581E+05
256x256x256	6,24136E+12	2,76643E+05

Tabla 12. Ejecución “MTP” para el modelo “Oldman”

En la se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Oldman”.

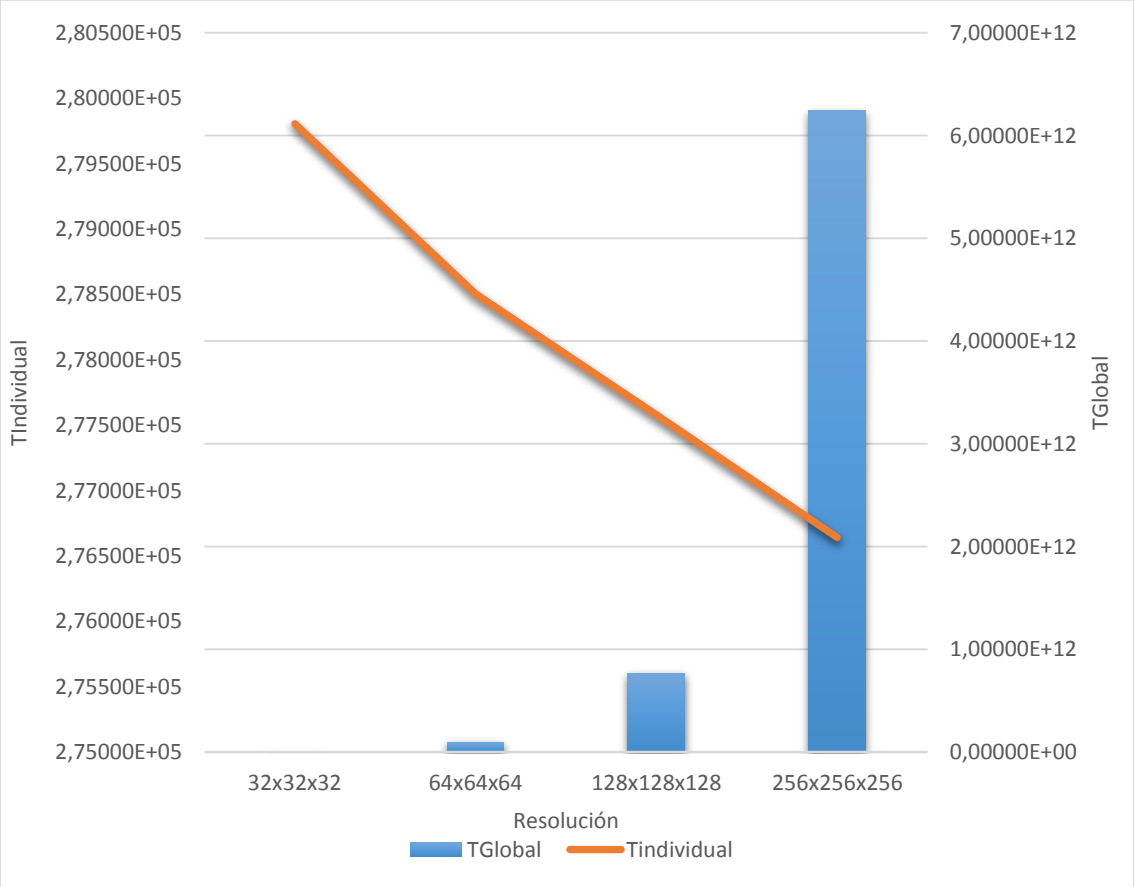

Figura 37. Gráfica de la ejecución “MTP” para el modelo “Oldman”

En la Tabla 13 se muestran los tiempos de ejecución para el modelo “Knife”.

Resolución	TGlobal	TIndividual
32x32x32	3,05705E+10	7,53636E+05
64x64x64	1,82003E+11	5,87583E+05
128x128x128	1,45818E+12	5,86960E+05
256x256x256	1,17056E+13	5,85496E+05

Tabla 13. Ejecución “MTP” para el modelo “Knife”

En la Figura 38 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Knife”.

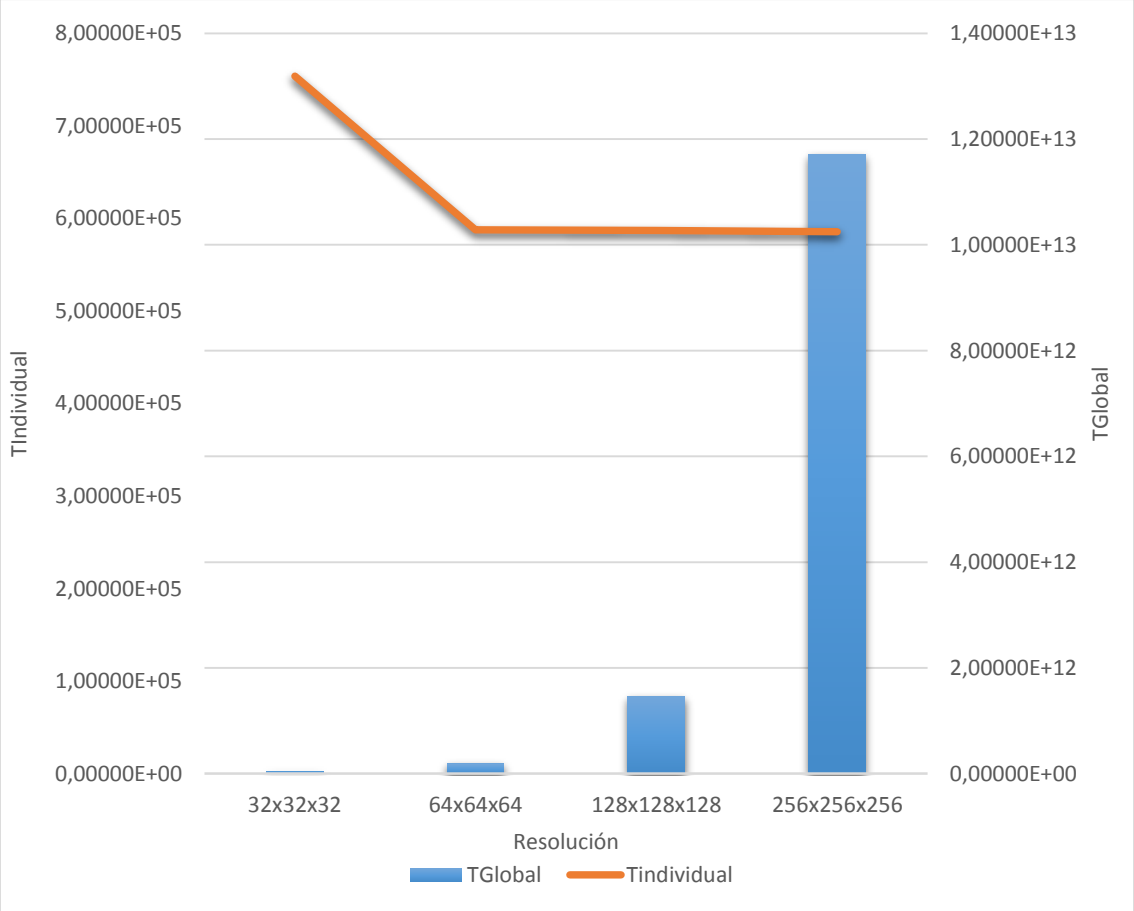

Figura 38. Gráfica de la ejecución “MTP” para el modelo “Knife”

4.6 EJECUCIONES DE “MTG”

Para poder interpretar los resultados de la ejecución “MTG” es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- El “TGlobal” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- El “TIndividual” se refiere a la media de ejecución de tiempo de la implementación por cada punto.

Cabe notar que en esta versión se han lanzado 4 hilos simultáneamente, dado que es el número de núcleos del procesador, sabiendo además que este no tiene Hyper-threading.

En la Tabla 14 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	1,62451E+09	4,95760E+04
64x64x64	1,29880E+10	4,95453E+04
128x128x128	1,03819E+11	4,95047E+04
256x256x256	8,29254E+11	4,94274E+04

Tabla 14. Ejecución “MTG” para el modelo “Apple”

En la Figura 39 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Knife”.

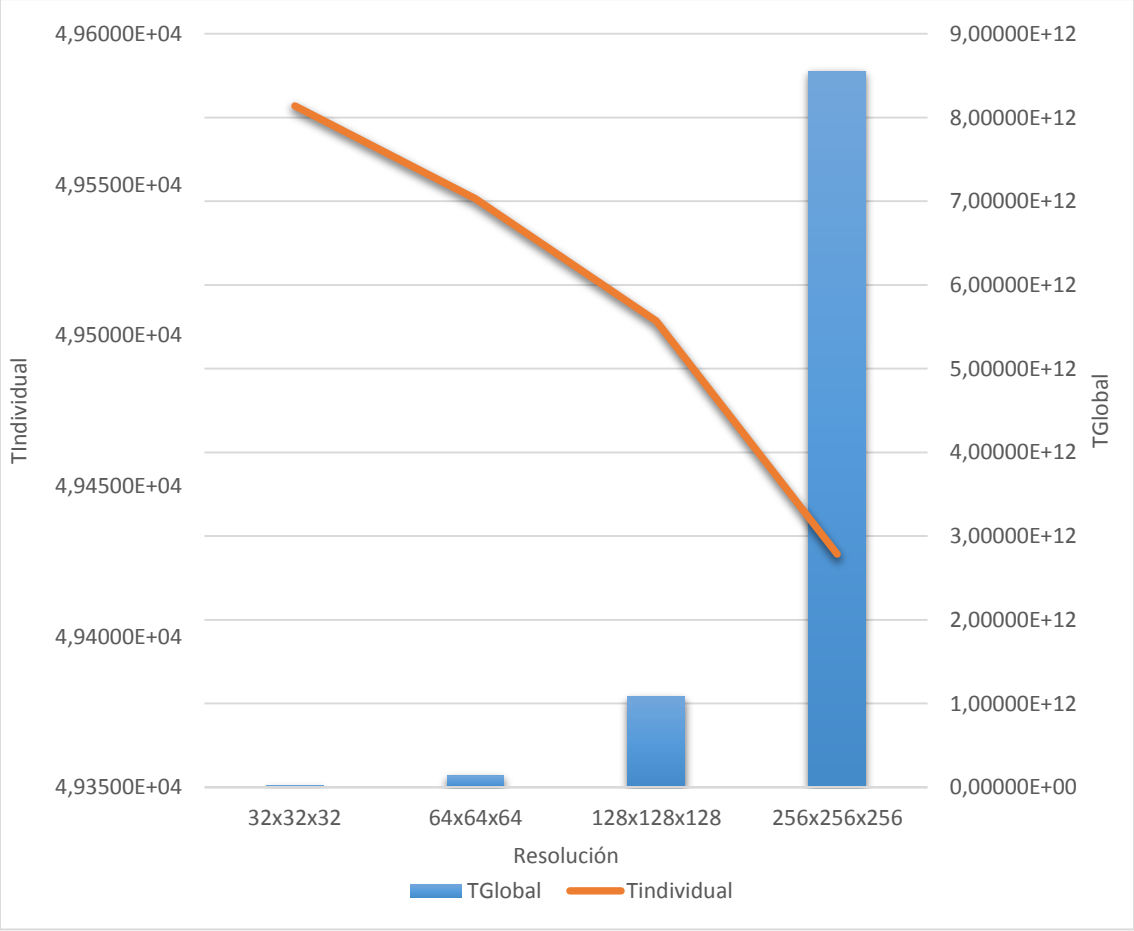

Figura 39. Gráfica de la ejecución “MTG” para el modelo “Apple”

En la Tabla 15 se muestran los tiempos de ejecución para el modelo “Oldman”.

Resolución	TGlobal	TIndividual
32x32x32	8,46536E+09	2,58342E+05
64x64x64	6,75540E+10	2,57698E+05
128x128x128	5,40277E+11	2,57624E+05
256x256x256	4,31248E+12	2,57044E+05

Tabla 15. Ejecución “MTG” para el modelo “Oldman”

En la Figura 40 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Oldman”.

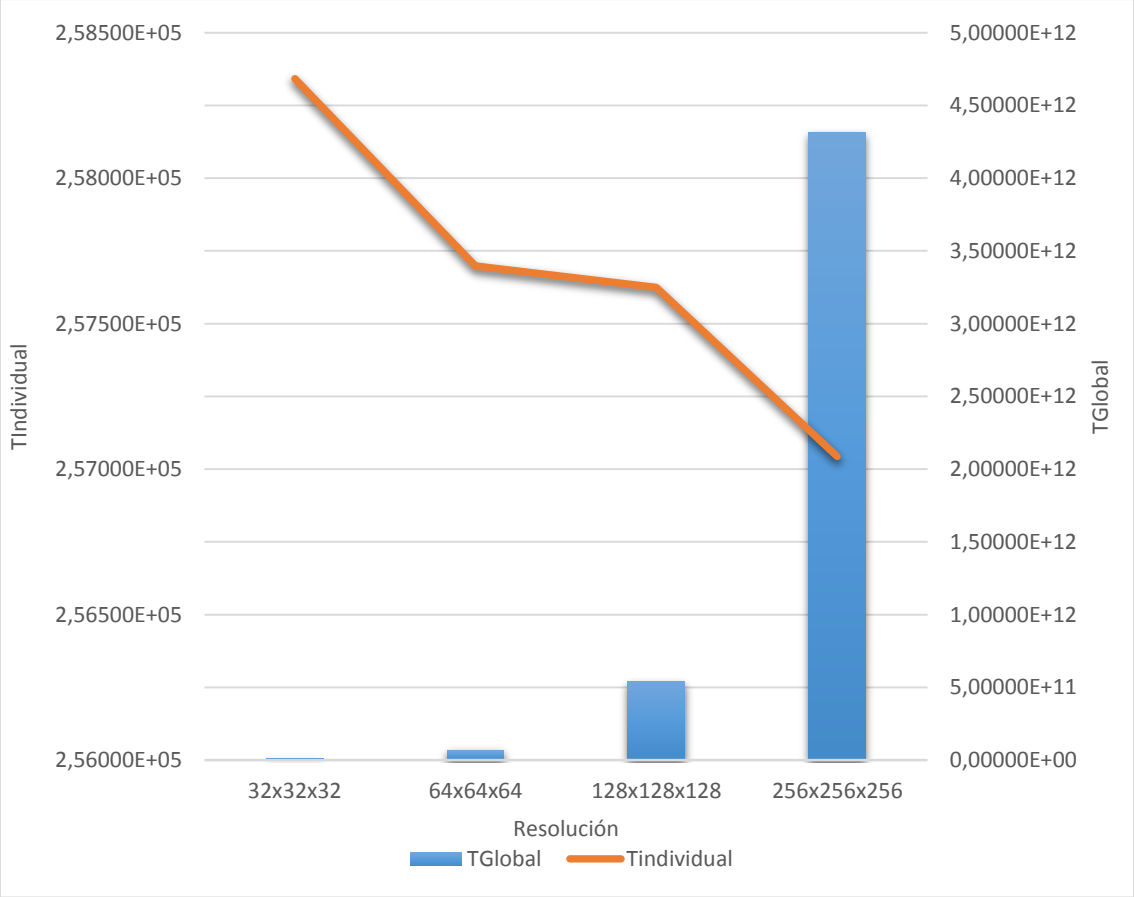

Figura 40. Gráfica de la ejecución “MTG” para el modelo “Oldman”

En la Tabla 16 se muestran los tiempos de ejecución para el modelo “Knife”.

Resolución	TGlobal	TIndividual
32x32x32	2,17042E+10	6,62361E+05
64x64x64	1,67911E+11	6,40531E+05
128x128x128	1,08002E+12	5,14994E+05
256x256x256	8,55394E+12	5,09855E+05

Tabla 16. Ejecución “MTG” para el modelo “Knife”

En la Figura 41 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Knife”.

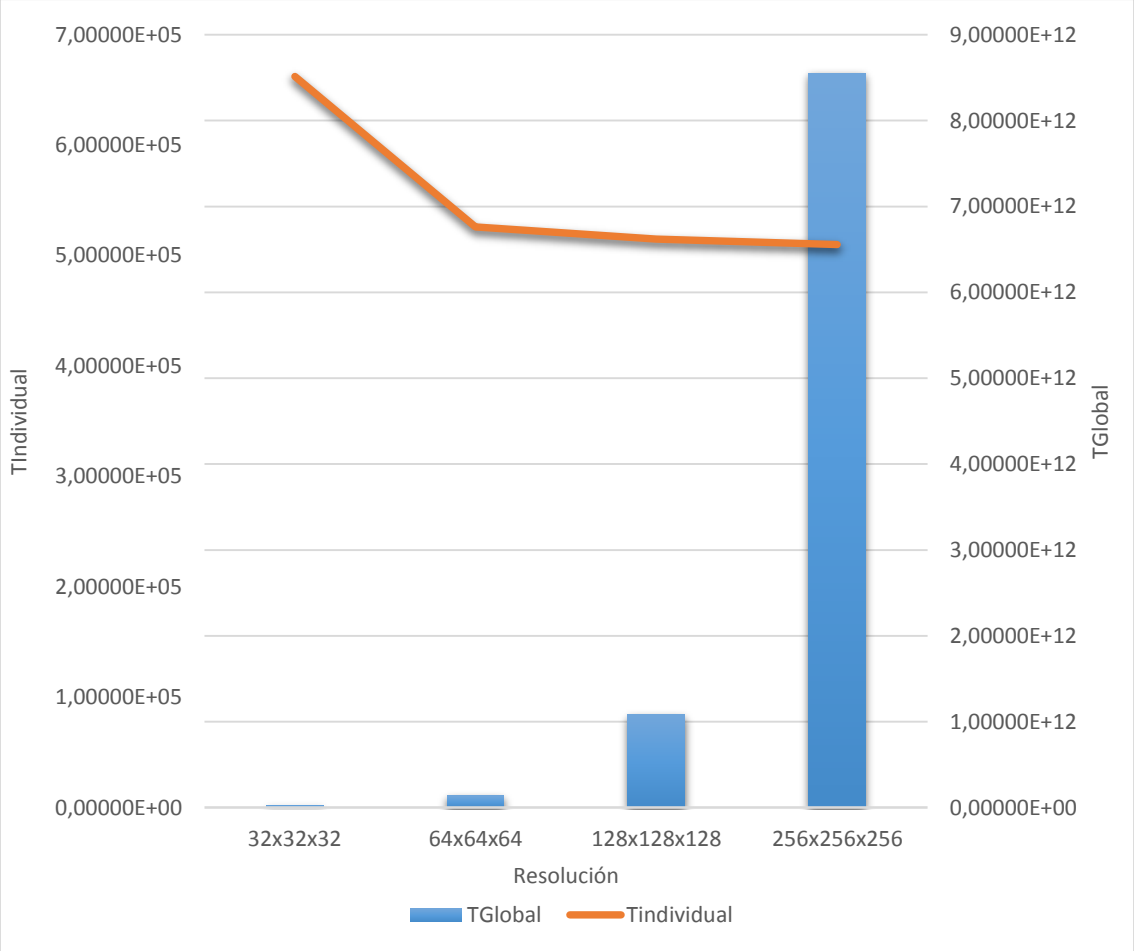

Figura 41. Gráfica de la ejecución “MTG” para el modelo “Knife”

4.7 EJECUCIONES DE “R660”

Para poder interpretar los resultados de la ejecución “R660” es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- El “TGlobal” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- El “TIndividual” se refiere a la media de ejecución de tiempo de la implementación por cada punto.

En la Tabla 17 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	1,19431E+09	3,64475E+04
64x64x64	8,56986E+09	3,26914E+04
128x128x128	6,83040E+10	3,25699E+04
256x256x256	5,45479E+11	3,25131E+04

Tabla 17. Ejecución “R660” para el modelo “Apple”

En la Figura 42 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Apple”.

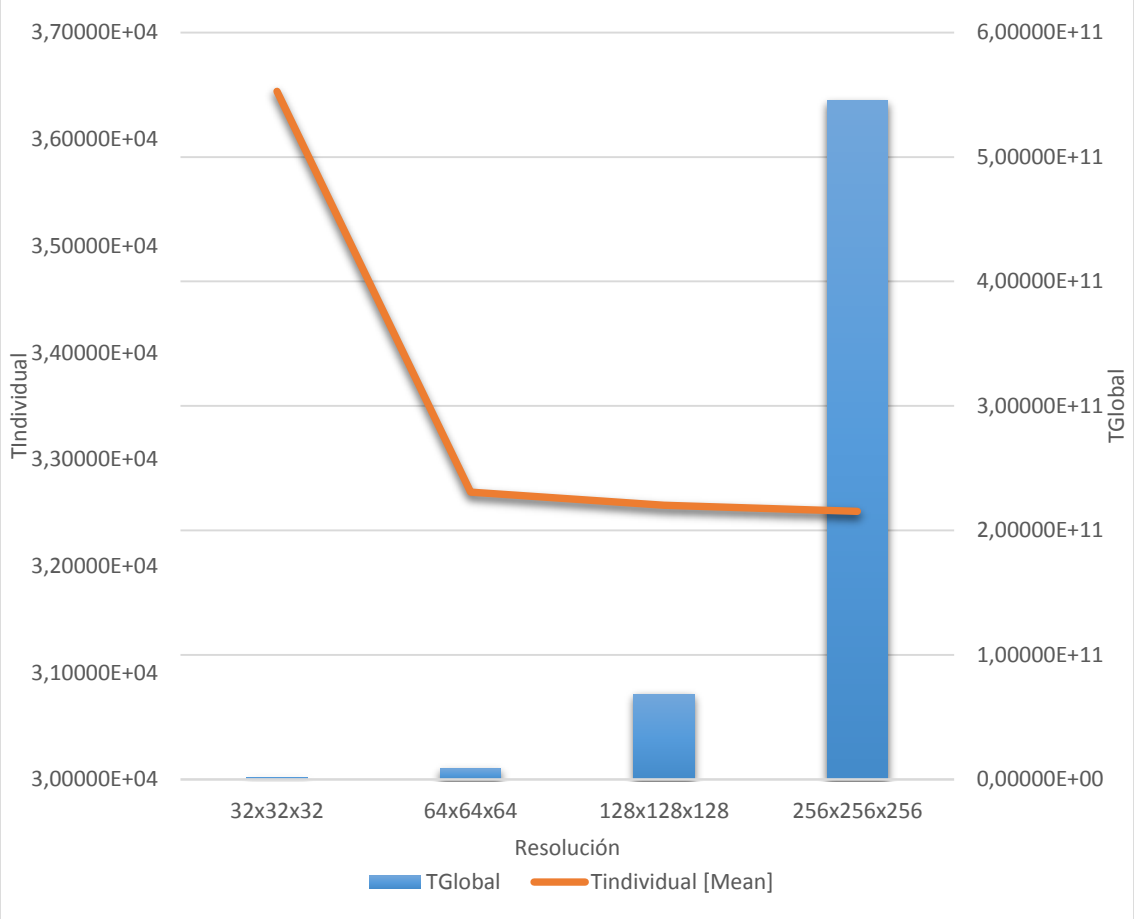

Figura 42. Gráfica de la ejecución “R660” para el modelo “Apple”

En la Tabla 18 se muestran los tiempos de ejecución para el modelo “Oldman”.

Resolución	TGlobal	TIndividual
32x32x32	4,25352E+09	1,29807E+05
64x64x64	3,28842E+10	1,25443E+05
128x128x128	2,62189E+11	1,25021E+05
256x256x256	2,05010E+12	1,22195E+05

Tabla 18. Ejecución “R660” para el modelo “Oldman”

En la Figura 43 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Oldman”.

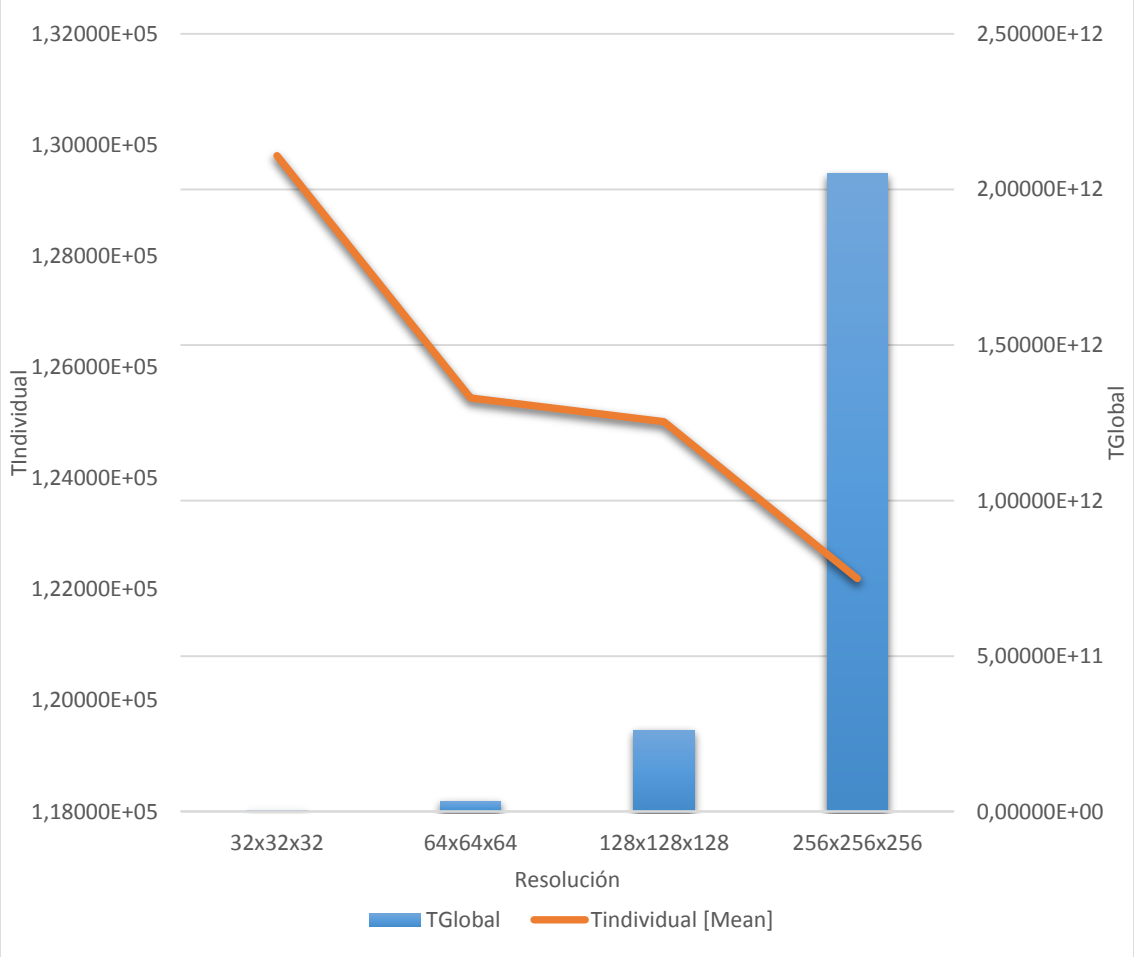

Figura 43. Gráfica de la ejecución “R660” para el modelo “Oldman”

En la Tabla 19 se muestran los tiempos de ejecución para el modelo “Knife”.

Resolución	TGlobal	TIndividual
32x32x32	7,46943E+09	2,27949E+05
64x64x64	5,87705E+10	2,24192E+05
128x128x128	4,64806E+11	2,21637E+05
256x256x256	3,71810E+12	2,21616E+05

Tabla 19. Ejecución “R660” para el modelo “Knife”

En la Figura 44 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Knife”.

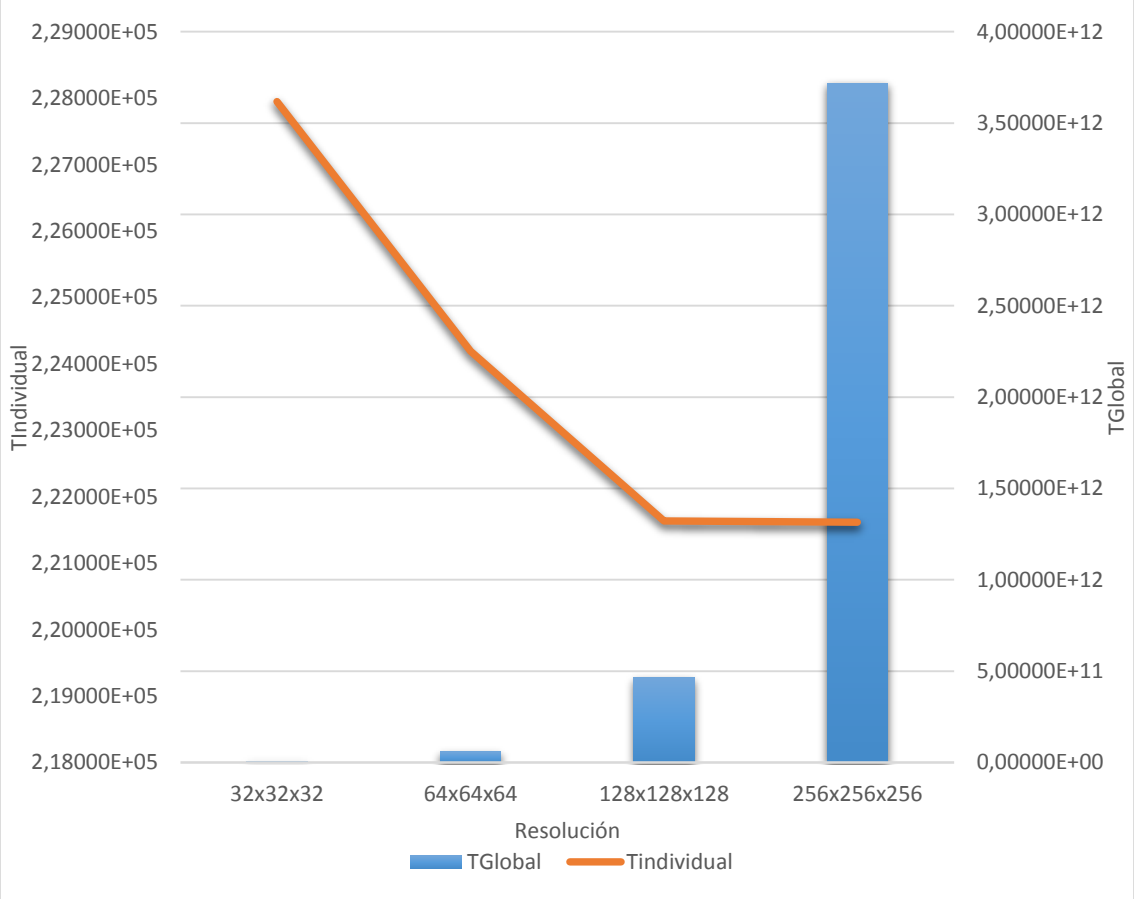

Figura 44. Gráfica de la ejecución “R660” para el modelo “Knife”

4.8 EJECUCIONES DE “G660”

Para poder interpretar los resultados de la ejecución “G660” es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- El “TGlobal” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- El “TIndividual” se refiere a la media de ejecución de tiempo de la implementación por cada punto.

En la Tabla 20 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	1,49920E+08	4,57519E+03
64x64x64	4,60204E+08	1,75554E+03
128x128x128	2,93348E+09	1,39879E+03
256x256x256	2,26819E+10	1,35194E+03

Tabla 20. Ejecución “G660” para el modelo “Apple”

En la Figura 45 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Apple”.

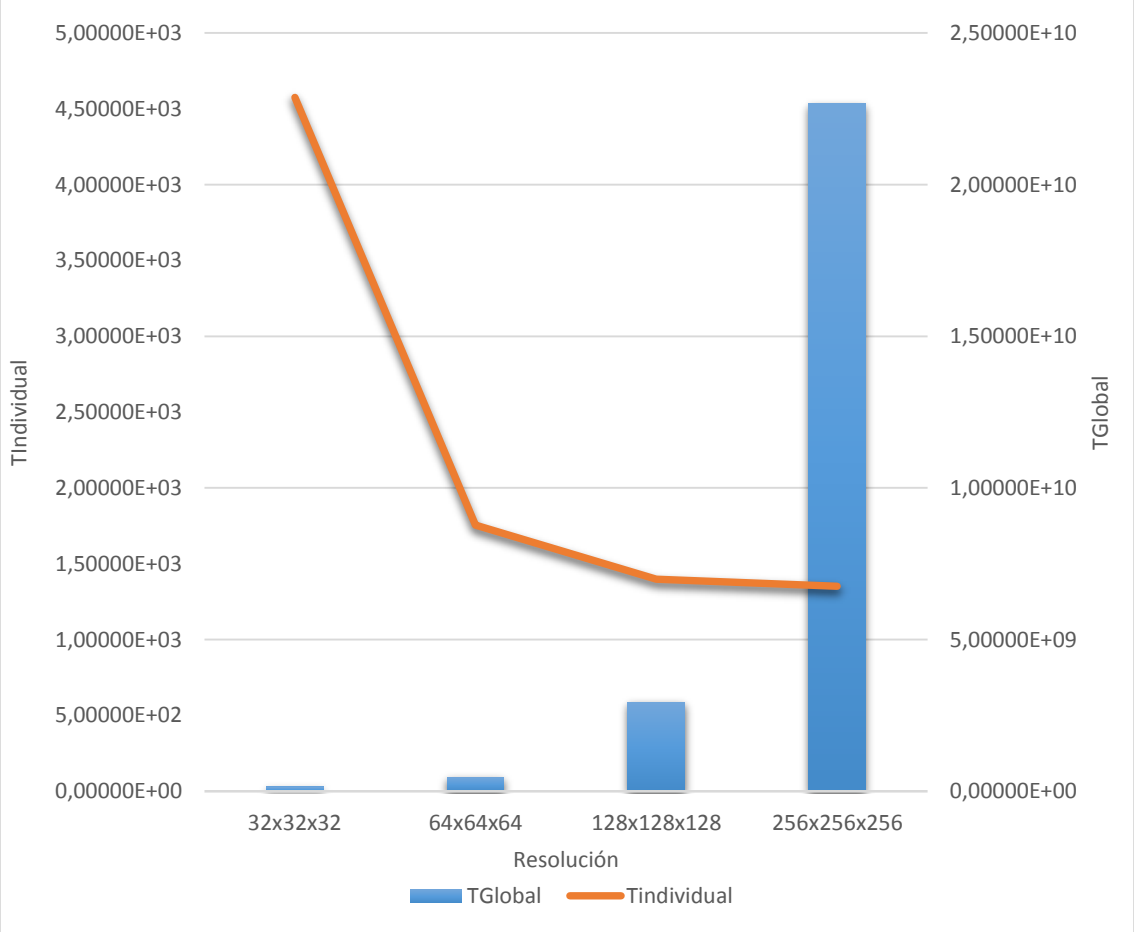

Figura 45. Gráfica de la ejecución “G660” para el modelo “Apple”

En la Tabla 21 se muestran los tiempos de ejecución para el modelo “Apple”.

Resolución	TGlobal	TIndividual
32x32x32	3,81629E+08	1,16464E+04
64x64x64	1,98607E+09	7,57627E+03
128x128x128	1,48662E+10	7,08874E+03
256x256x256	1,16320E+11	6,93324E+03

Tabla 21. Ejecución “G660” para el modelo “Oldman”

En la Figura 46 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Oldman”.

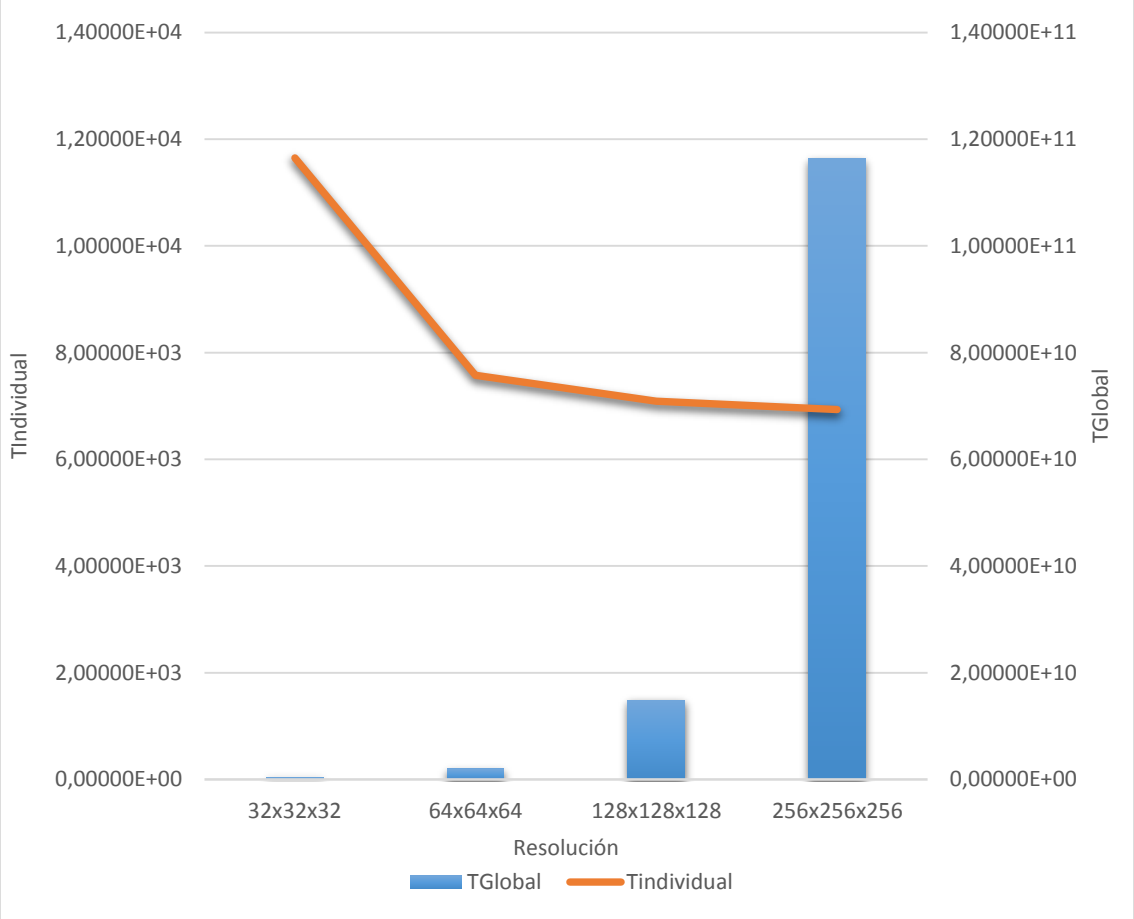

Figura 46. Gráfica de la ejecución “G660” para el modelo “Oldman”

En la Tabla 22 se muestran los tiempos de ejecución para el modelo “Knife”.

Resolución	TGlobal	TIndividual
32x32x32	7,88533E+08	2,40641E+04
64x64x64	3,79774E+09	1,44872E+04
128x128x128	2,90880E+10	1,38703E+04
256x256x256	2,24609E+11	1,33878E+04

Tabla 22. Ejecución “G660” para el modelo “Knife”

En la Figura 47 se muestra una representación gráfica tanto del tiempo global como del tiempo individual de las distintas ejecuciones para el modelo “Knife”.

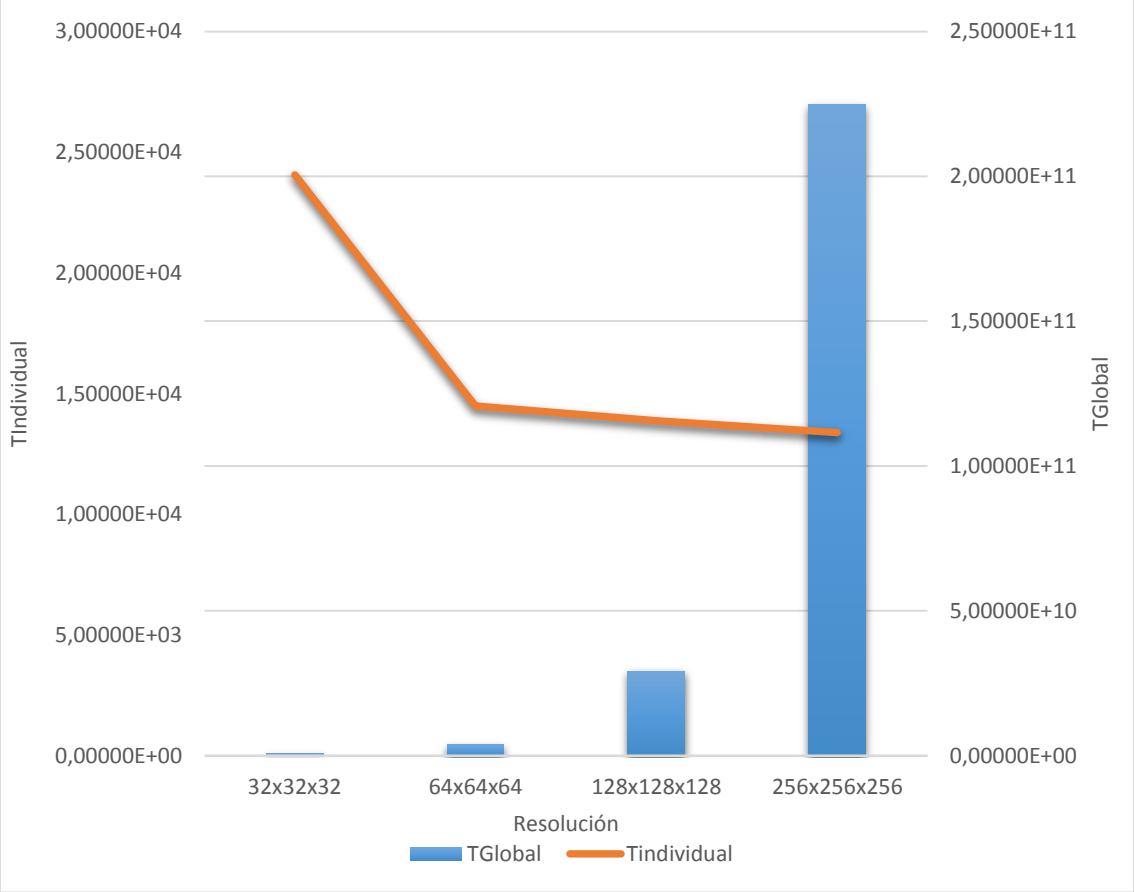

Figura 47. Gráfica de la ejecución “G660” para el modelo “Knife”

4.9 COMPARACIONES POINT-POLY Y GRID-POLY

Como ya he mencionado con anterioridad, he desarrollado dos versiones del algoritmo para cada paradigma de programación.

Sin embargo, como se puede observar, y como veremos a continuación las versiones que evalúan un conjunto de puntos, tienden a dar mejores resultados con respecto a las versiones que evalúan los puntos individualmente

Para poder interpretar los resultados de las comparaciones es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- “Point-Poly” indica el tiempo que se invierte en evaluar un único punto para la versión que evalúa un solo punto.
- “Grid-Poly” indica el tiempo que se invierte en evaluar un único punto para la versión que evalúa un conjunto de puntos.
- “Ganancia” es el factor de la versión “Grid-Poly” con respecto a la versión “Point-Poly”.

4.9.1 “STP” comparado con “STG”

En la Tabla 23 se muestran la comparación para el modelo “Apple”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	1,78327E+05	1,77869E+05	1,00257458413
64x64x64	1,78213E+05	1,77629E+05	1,00328564103
128x128x128	1,77654E+05	1,77596E+05	1,00033030410
256x256x256	1,77474E+05	1,77337E+05	1,00076836852

Tabla 23. Comparación “STP” vs “STG” para “Apple”

En la Tabla 24 se muestran la comparación para el modelo “Oldman”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	9,29581E+05	9,27162E+05	1,00260867342
64x64x64	9,27876E+05	9,26078E+05	1,00194160299
128x128x128	9,26243E+05	9,25868E+05	1,00040452674
256x256x256	9,24422E+05	9,23686E+05	1,00079696884

Tabla 24. Comparación “STP” vs “STG” para “Oldman”

En la Tabla 25 se muestran la comparación para el modelo “Knife”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	2,12043E+06	2,08843E+06	1,01531944381
64x64x64	2,06600E+06	2,05485E+06	1,00542685557
128x128x128	1,90964E+06	1,90865E+06	1,00051793343
256x256x256	1,90589E+06	1,90138E+06	1,00237043920

Tabla 25. Comparación “STP” vs “STG” para “Knife”

En la Figura 48 se muestra una representación gráfica de las ganancias de “STG” con respecto de “STP” para todos los modelos.

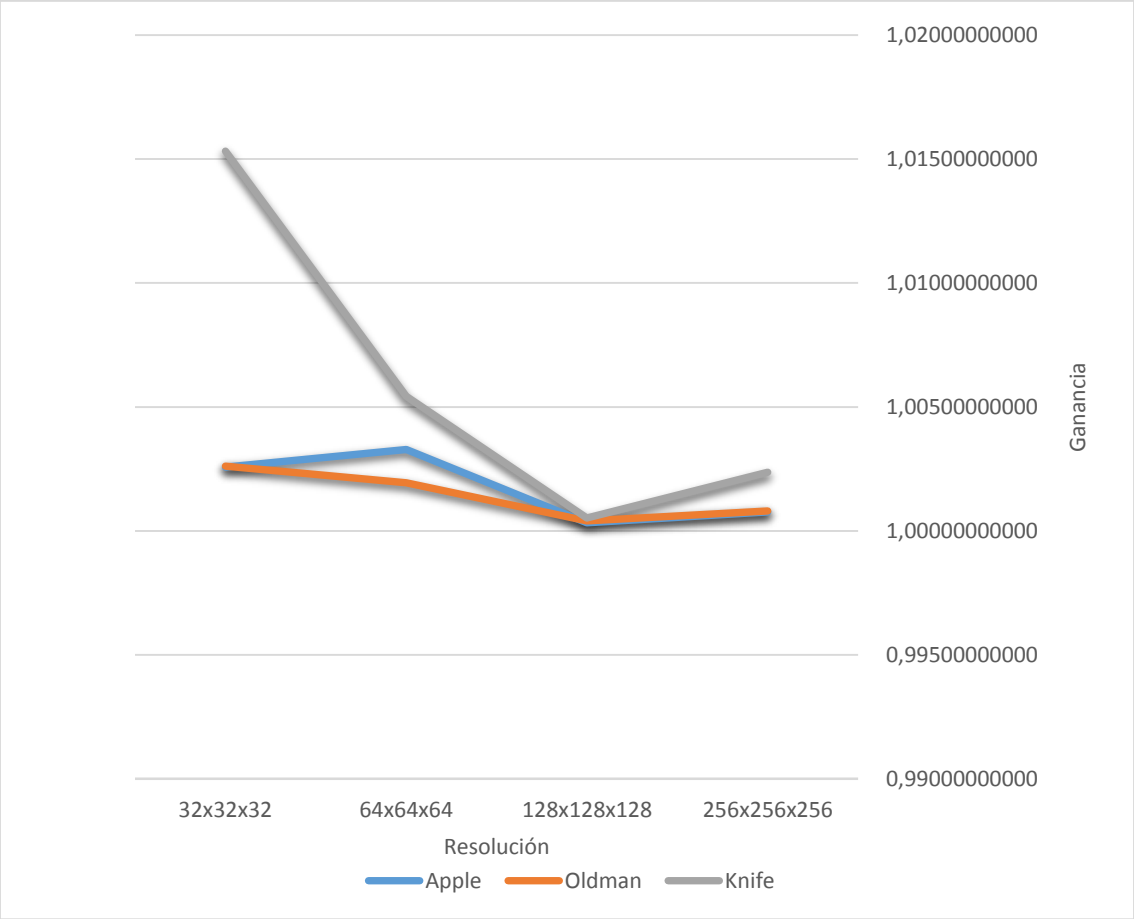

Figura 48. Comparación de las ganancias “STG” vs “STP” (1)

Tal y como se puede observar en las versiones “STP” y “STG” no hay prácticamente no hay ganancia alguna. Esto se debe a que, aunque la versión “STP” realiza un número alto de llamadas en comparación con la versión “STG”, la arquitectura de CPU está altamente optimizada para manejar el “branching”. Por tanto no se ve reflejado en alta medida el tiempo necesario para evaluar cada uno de los puntos de manera individual, en vez de evaluarlos todos ellos en la misma llamada.

Por otro lado también es cierto, que no existe el problema de la necesidad de transferencia de los datos a la memoria, pues el acceso desde CPU es directo a memoria, mediante el controlador de memoria.

4.9.2 “MTP” comparado con “MTG”

En la Tabla 26 se muestran la comparación para el modelo “Apple”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	6,00547E+04	4,95760E+04	1,21136606554
64x64x64	5,94534E+04	4,95453E+04	1,19998029693
128x128x128	5,94369E+04	4,95047E+04	1,20063149934
256x256x256	5,93462E+04	4,94274E+04	1,20067406617

Tabla 26. Comparación “MTP” vs “MTG” para “Apple”

En la Tabla 27 se muestran la comparación para el modelo “Oldman”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	2,79806E+05	2,58342E+05	1,08308127004
64x64x64	2,78505E+05	2,57698E+05	1,08074253723
128x128x128	2,77581E+05	2,57624E+05	1,07746316404
256x256x256	2,76643E+05	2,57044E+05	1,07624766324

Tabla 27. Comparación “MTP” vs “MTG” para “Oldman”

En la Tabla 28 se muestran la comparación para el modelo “Knife”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	7,53636E+05	6,62361E+05	1,13780259959
64x64x64	5,87583E+05	5,26090E+05	1,11688680461
128x128x128	5,86960E+05	5,14994E+05	1,13974123631
256x256x256	5,85496E+05	5,09855E+05	1,14835941544

Tabla 28. Comparación “MTP” vs “MTG” para “Knife”

En la Figura 49 se muestra una representación gráfica de las ganancias de “MTG” con respecto de “MTP” para todos los modelos.

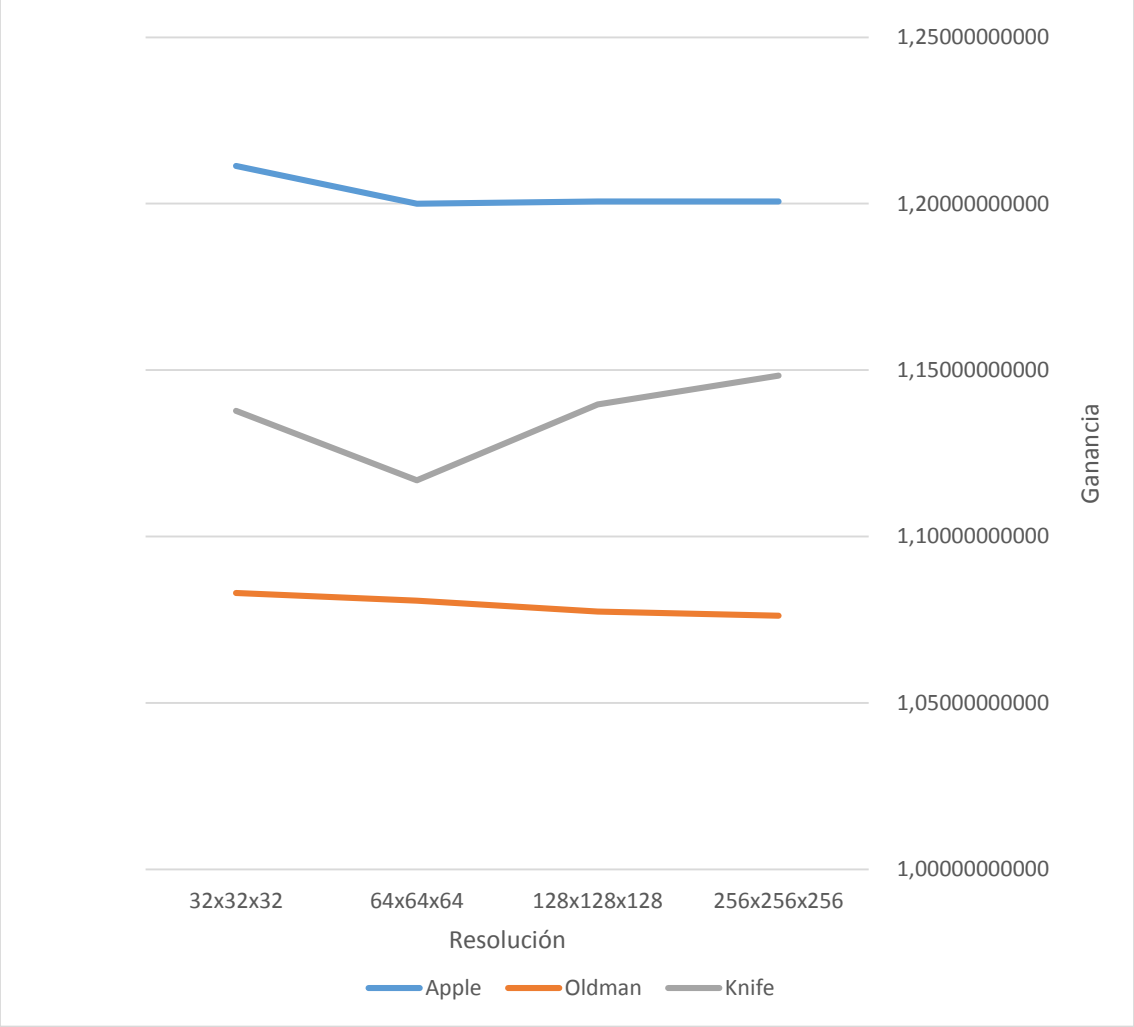

Figura 49. Comparación de las ganancias “MTG” vs “MTP”

En este caso la diferencia del tiempo de ejecución entre “MTG” y “MTP” se refleja en una ganancia ligeramente superior con respecto a las versiones monohilo del procesador. Esto es debido a que el proceso de crear los hilos, supone una notable parte del tiempo de ejecución para ambos algoritmos, y dado que la versión “MTP” crea muchos más hilos que la versión “MTG”, esto resulta en tiempos de ejecución mejores para “MTG”.

4.9.3 “R660” comparado con “G660”

En la Tabla 29 se muestran la comparación para el modelo “Apple”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	3,64475E+04	4,57519E+03	7,96633575944
64x64x64	3,26914E+04	1,75554E+03	18,62189489729
128x128x128	3,25699E+04	1,39879E+03	23,28429443225
256x256x256	3,25131E+04	1,35194E+03	24,04910310902

Tabla 29. Comparación “R660” vs “R660” para “Apple”

En la Tabla 30 se muestran la comparación para el modelo “Oldman”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	1,29807E+05	1,16464E+04	11,14569135499
64x64x64	1,25443E+05	7,57627E+03	16,55739281084
128x128x128	1,25021E+05	7,08874E+03	17,63661163419
256x256x256	1,22195E+05	6,93324E+03	17,62455183390

Tabla 30. Comparación “R660” vs “G660” para “Oldman”

En la Tabla 31 se muestran la comparación para el modelo “Knife”.

Resolución	Point-Poly	Grid-Poly	Ganancia
32x32x32	2,27949E+05	2,40641E+04	9,47256670005
64x64x64	2,24192E+05	1,44872E+04	15,47513422952
128x128x128	2,21637E+05	1,38703E+04	15,97926719856
256x256x256	2,21616E+05	1,33878E+04	16,55362705990

Tabla 31. Comparación “R660” vs “G660” para “Knife”

En la Figura 50 se muestra una representación gráfica de las ganancias de “MTG” con respecto de “MTP” para todos los modelos.

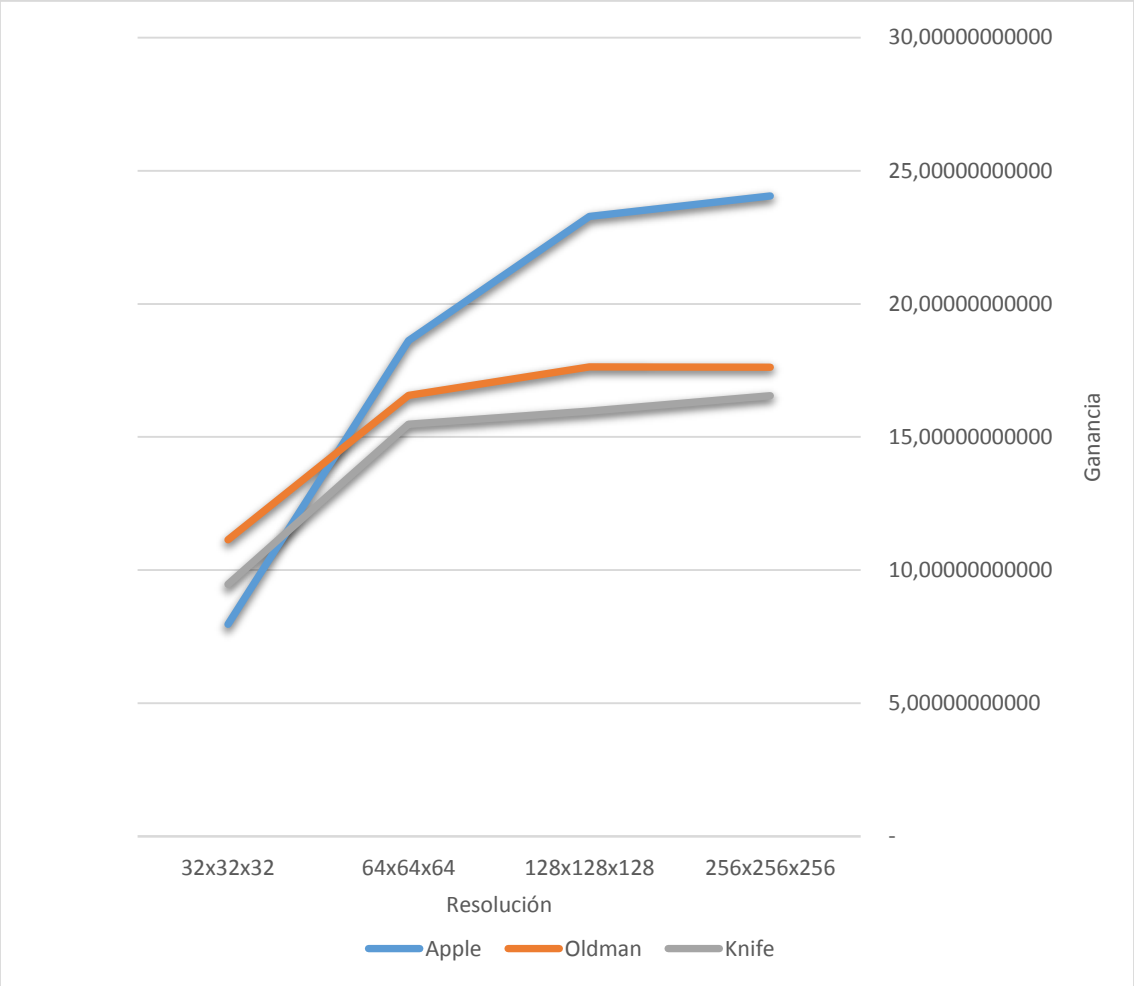

Figura 50. Comparación de las ganancias “R660” vs “G660”

Al comparar las versiones de GPU, se observa una gran diferencia en el tiempo de ejecución entre “R660” y “G660”. Esto se debe a que lanzar núcleos para ejecutar en la GPU supone un gran cuanto de tiempo, siendo esto consecuencia de la forma en que funciona la arquitectura de CUDA. Por otro lado es más rentable hacer pocas transferencias de memoria grandes que muchas transferencias pequeñas.

4.10 COMPARACIONES CON “STP”

Es una práctica común el comparar los nuevos algoritmos que hacen uso de paralelización con su equivalente versión en un solo flujo de ejecución. Esto es lo que trato de abordar en esta sección, con el mero objetivo de demostrar lo rentable que es introducir paralelismo a tu algoritmo, siempre que sea posible.

Como es obvio no se incluye la información referente a la versión “STP”, pues es la utilizada para comparar. En caso de querer ver la referencia para la ganancia, los tiempos de ejecución de “STP” se muestran en la Tabla 5, la Tabla 6 y la Tabla 7.

Por otro lado, en esta sección no incluyo la información referente a la comparación con la versión “STG”, pues dicha comparación ya se realiza en el apartado 4.9.1. Además no aporta información alguna sobre la mejora en paralelismo, ya que esta versión no utiliza el paralelismo, el cual es el objeto de estudio en esta sección.

Para poder interpretar los resultados de las comparaciones es necesario tener en cuenta lo siguiente:

- Las unidades de tiempo están expresadas en nanosegundos.
- La “Resolución” indica el tamaño del conjunto de puntos, siendo el producto de cada uno de las dimensiones el número de puntos evaluados.
- “Tiempo” indica el tiempo total que se ha invertido en evaluar todos los puntos.
- “Ganancia” es la relación del tiempo del algoritmo evaluado con respecto a la versión “STP”.

4.10.1 “MTP” comparado con “STP”

En la Tabla 32 se muestran la comparación para el modelo “Apple”.

Resolución	Tiempo	Ganancia
32x32x32	4,77167E+09	1,22460707739
64x64x64	3,79092E+10	1,23234869765
128x128x128	3,04425E+11	1,22383981008
256x256x256	2,50575E+12	1,18827286142

Tabla 32. Ganancia de “MTP” con “STP” para “Apple”

En la Tabla 33 se muestran la comparación para el modelo “Oldman”.

Resolución	Tiempo	Ganancia
32x32x32	1,19555E+10	2,54782297003
64x64x64	9,65888E+10	2,51827506656
128x128x128	7,65960E+11	2,53599561456
256x256x256	6,24136E+12	2,48491207961

Tabla 33. Ganancia de “MTP” con “STP” para “Oldman”

En la Tabla 34 se muestran la comparación para el modelo “Knife”.

Resolución	Tiempo	Ganancia
32x32x32	3,05705E+10	2,23855621066
64x64x64	1,82003E+11	2,95964501187
128x128x128	1,45818E+12	2,74503236021
256x256x256	1,17056E+13	2,72518546967

Tabla 34. Ganancia de “MTP” con “STP” para “Knife”

En la Figura 51 se muestra una representación gráfica de las ganancias de “MTP” con respecto a la versión “STP” para todos los modelos.

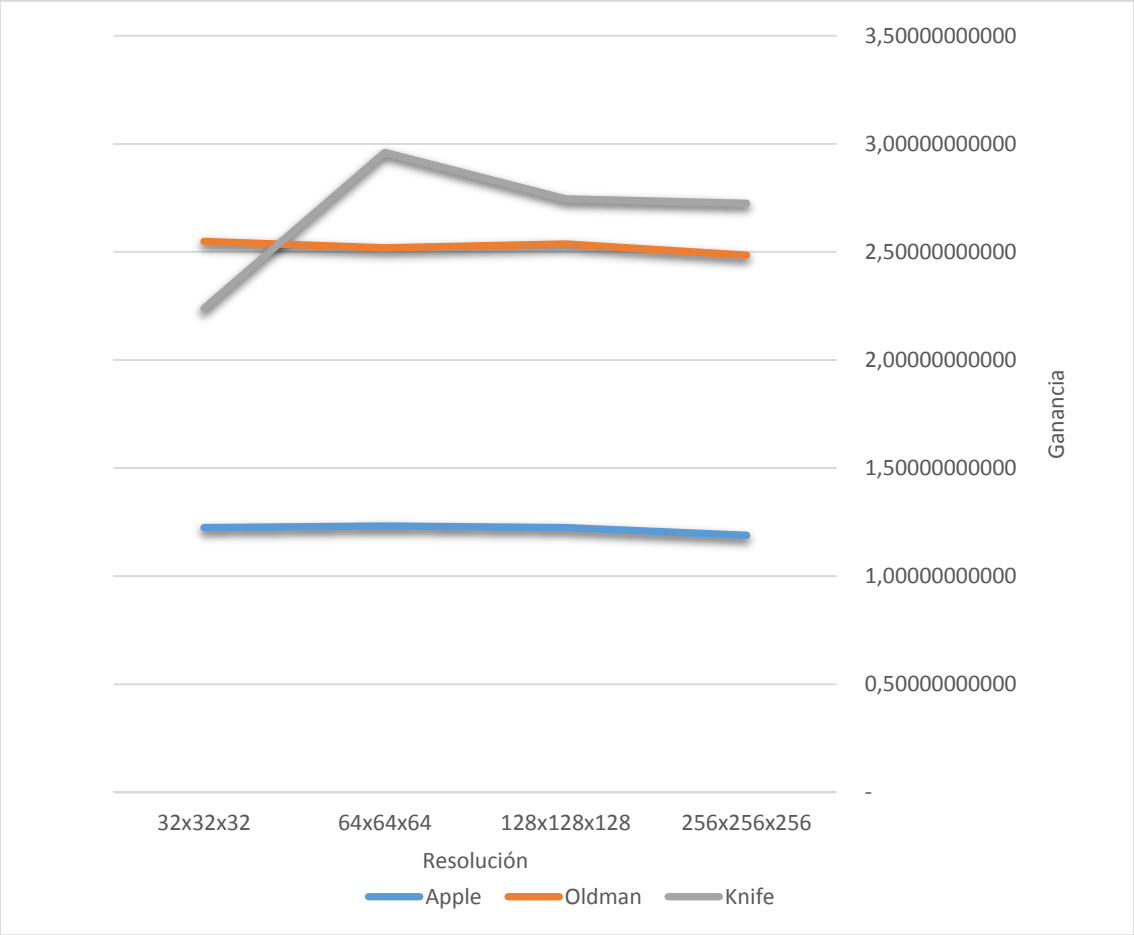

Figura 51. Comparación de las ganancias “MTP” vs “STP”

A partir de esta comparación se comprueba como el simple hecho de hacer uso de la funcionalidad multi-núcleo de los procesadores actuales, suponen una gran ventaja en tiempo de computación para aquellos algoritmos que se puedan ejecutar de forma paralela.

Sin embargo, aun habiendo lanzado 4 hilos de manera simultánea, los resultados no se acercan lo suficiente al factor 4 de ganancia. Debiéndose esto a que la implementación “MTP” lanza esos cuatros hilos por cada punto que se evalúa, con lo que se pierde bastante tiempo manejando la ejecución de dichos hilos.

4.10.2 “MTG” comparado con “STP”

En la Tabla 35 se muestran la comparación para el modelo “Apple”.

Resolución	Tiempo	Ganancia
32x32x32	1,62451E+09	3,59704026005
64x64x64	1,29880E+10	3,59696385239
128x128x128	1,03819E+11	3,58863457180
256x256x256	8,29254E+11	3,59059474451

Tabla 35. Ganancia de “MTG” con “STP” para “Apple”

En la Tabla 36 se muestran la comparación para el modelo “Oldman”.

Resolución	Tiempo	Ganancia
32x32x32	8,46536E+09	3,59825354760
64x64x64	6,75540E+10	3,60063018640
128x128x128	5,40277E+11	3,59532299195
256x256x256	4,31248E+12	3,59636049548

Tabla 36. Ganancia de “MTG” con “STP” para “Oldman”

En la Tabla 37 se muestran la comparación para el modelo “Knife”.

Resolución	Tiempo	Ganancia
32x32x32	2,17042E+10	3,15301570378
64x64x64	1,37911E+11	3,90588555277
128x128x128	1,08002E+12	3,70616782046
256x256x256	8,55394E+12	3,72926769152

Tabla 37. Ganancia de “MTG” con “STP” para “Knife”

En la Figura 52 se muestra una representación gráfica de las ganancias de “MTG” con respecto a la versión “STP” para todos los modelos.

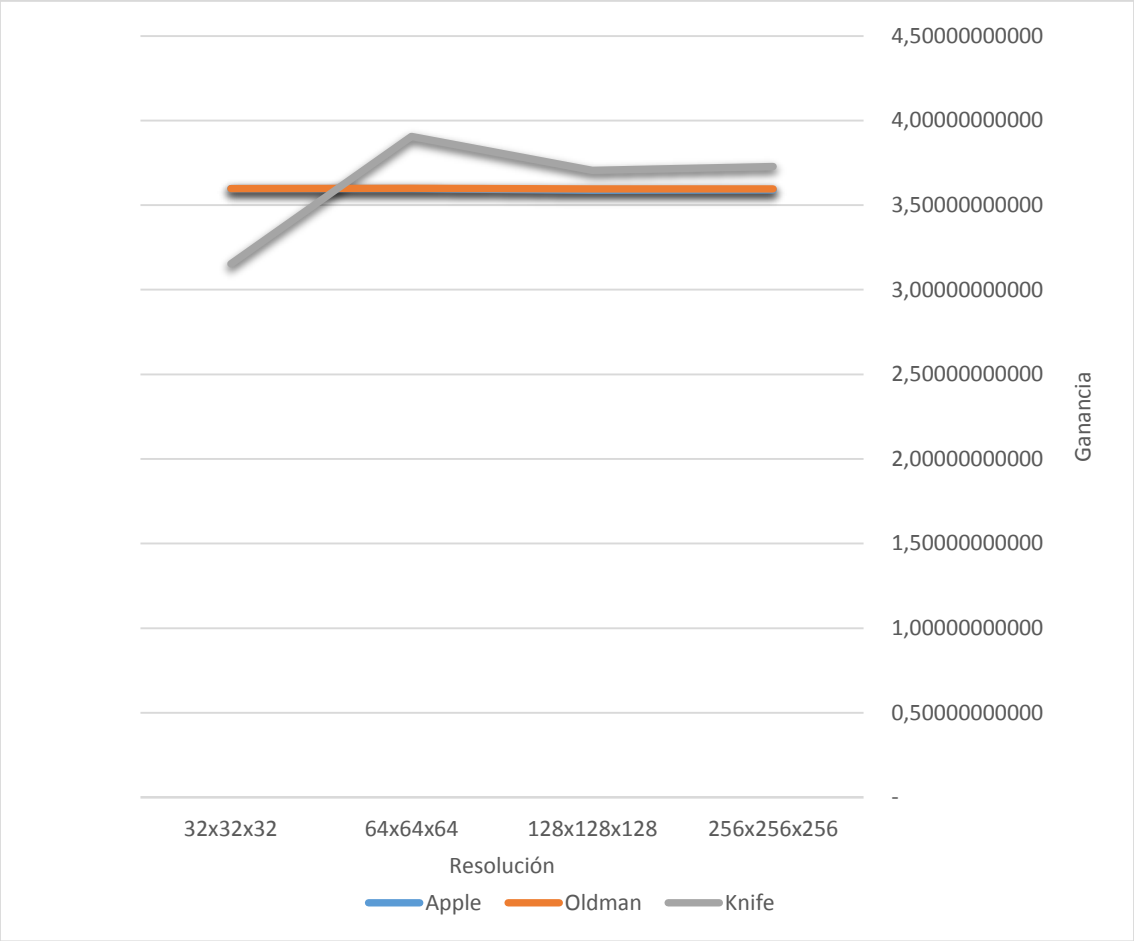

Figura 52. Comparación de las ganancias “MTP” vs “STP”

En este caso, la versión “MTG” muestra una alta ganancia, cercana a 4. Esto se debe a que a diferencia de la versión “MTP”, la versión “MTG” solo lanza 4 hilos, de tal modo que el tiempo en la gestión de los hilos es ínfimo comparado con el tiempo de procesamiento.

4.10.3 “R660” comparado con “STP”

En la Tabla 38 se muestran la comparación para el modelo “Apple”.

Resolución	Tiempo	Ganancia
32x32x32	1,19431E+09	4,89271422538
64x64x64	8,56986E+09	5,45135519567
128x128x128	6,83040E+10	5,45455200084
256x256x256	5,45479E+11	5,45853465174

Tabla 38. Ganancia de “R660” con “STP” para “Apple”

En la Tabla 39 se muestran la comparación para el modelo “Oldman”.

Resolución	Tiempo	Ganancia
32x32x32	4,25352E+09	7,16124355100
64x64x64	3,28842E+10	7,39677639012
128x128x128	2,62189E+11	7,40867822921
256x256x256	2,05010E+12	7,56512379057

Tabla 39. Ganancia de “R660” con “STP” para “Oldman”

En la Tabla 40 se muestran la comparación para el modelo “Knife”.

Resolución	Tiempo	Ganancia
32x32x32	7,46943E+09	9,16184501689
64x64x64	5,87705E+10	9,16557511257
128x128x128	4,64806E+11	8,61164040772
256x256x256	3,71810E+12	8,57964219985

Tabla 40. Ganancia de “R660” con “STP” para “Knife”

En la Figura 53 se muestra una representación gráfica de las ganancias de “R660” con respecto a la versión “STP” para todos los modelos.

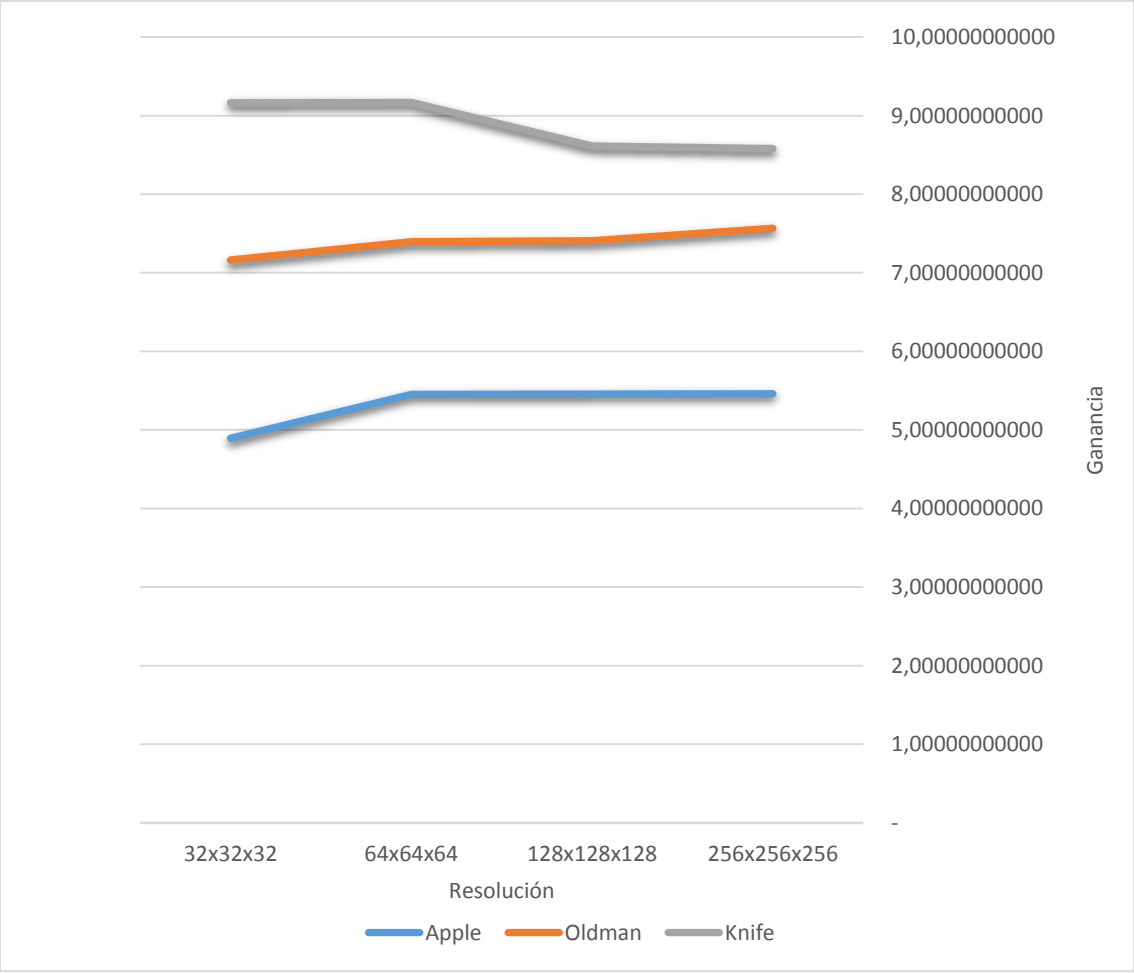

Figura 53. Comparación de las ganancias “R660” vs “STP”

En este caso la versión de CUDA que emplea reducción muestra unas ganancias más altas que “MTP” y “MTG”, ya que se están ejecutando sobre la tarjeta gráfica.

Sin embargo, no se explota todo el potencial del procesador gráfico debido a que se evalúa cada punto de forma individual. Suponiendo esto que se dedique un alto cuanto de tiempo al lanzar los núcleos.

4.10.4 “G660” comparado con “STP”

En la Tabla 41 se muestran la comparación para el modelo “Apple”.

Resolución	Tiempo	Ganancia
32x32x32	1,49920E+08	38,97700429431
64x64x64	4,60204E+08	101,51456350162
128x128x128	2,93348E+09	127,00539478352
256x256x256	2,26819E+10	131,27286266381

Tabla 41. Ganancia de “G660” con “STP” para “Apple”

En la Tabla 42 se muestran la comparación para el modelo “Oldman”.

Resolución	Tiempo	Ganancia
32x32x32	3,81629E+08	79,81701033733
64x64x64	1,98607E+09	122,47133222520
128x128x128	1,48662E+10	130,66398065127
256x256x256	1,16320E+11	133,33191637675

Tabla 42. Ganancia de “G660” con “STP” para “Oldman”

En la Tabla 43 se muestran la comparación para el modelo “Knife”.

Resolución	Tiempo	Ganancia
32x32x32	7,88533E+08	86,78618801802
64x64x64	3,79774E+09	141,83850515776
128x128x128	2,90880E+10	137,60770309288
256x256x256	2,24609E+11	142,02419728358

Tabla 43. Ganancia de “G660” con “STP” para “Knife”

En la se muestra una representación gráfica de las ganancias de “G660” con respecto a la versión “STP” para todos los modelos.

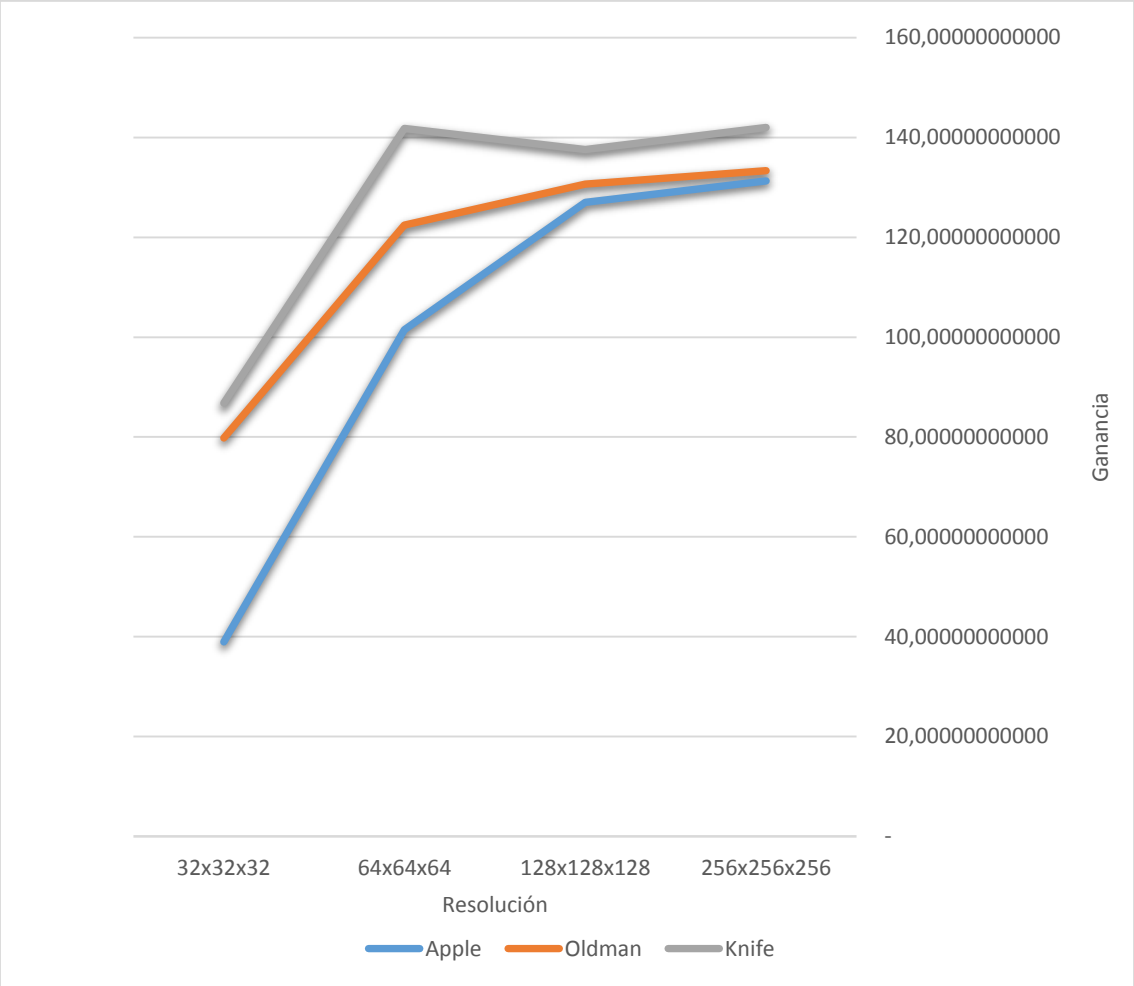

Figura 54. Comparación de las ganancias “G660” vs “STP”

Tal y como se puede observar en la versión “G660” aumenta drásticamente la ganancia, pues únicamente se ejecuta un núcleo maximizando el rendimiento del procesador gráfico.

Es importante notar que en esta versión cuanto mayor sea el modelo, así como la cantidad de puntos evaluados, mayor es la ganancia obtenida. Significando esto que cuanto más procesamiento se haga más rentable es el uso de esta versión.

4.11 VOXELIZACIÓN CSG

La renderización de modelos CSG cuyas primitivas son B-Rep voxelizados, es una de las aplicaciones que ya he descrito con anterioridad.

En esta sección se muestran 3 ejemplos de voxelización CSG. En cada uno de estos se muestra la operación CSG de la cual resultan, estando plasmada esta mediante una expresión RPN.

Para interpretar la expresión RPN tal y como yo la he plasmado, es necesario conocer que se marcan las primitivas con un identificador numérico empezando desde el 0. En lo que a las operaciones booleanas respecta, se representa la intersección con una “I”, la unión con una “U” y la diferencia con una “D”.

Por ejemplo, si tuviésemos un modelo CSG resultante de operar con 4 primitivas y la siguiente expresión RPN:

0 2 3 I 1 D U

Se realizaría la intersección de la tercera y cuarta primitiva, a esto se le calcularía la diferencia con la segunda primitiva, y a esto último se le realizaría la unión con la primera primitiva.

4.11.1 Modelo de pistón

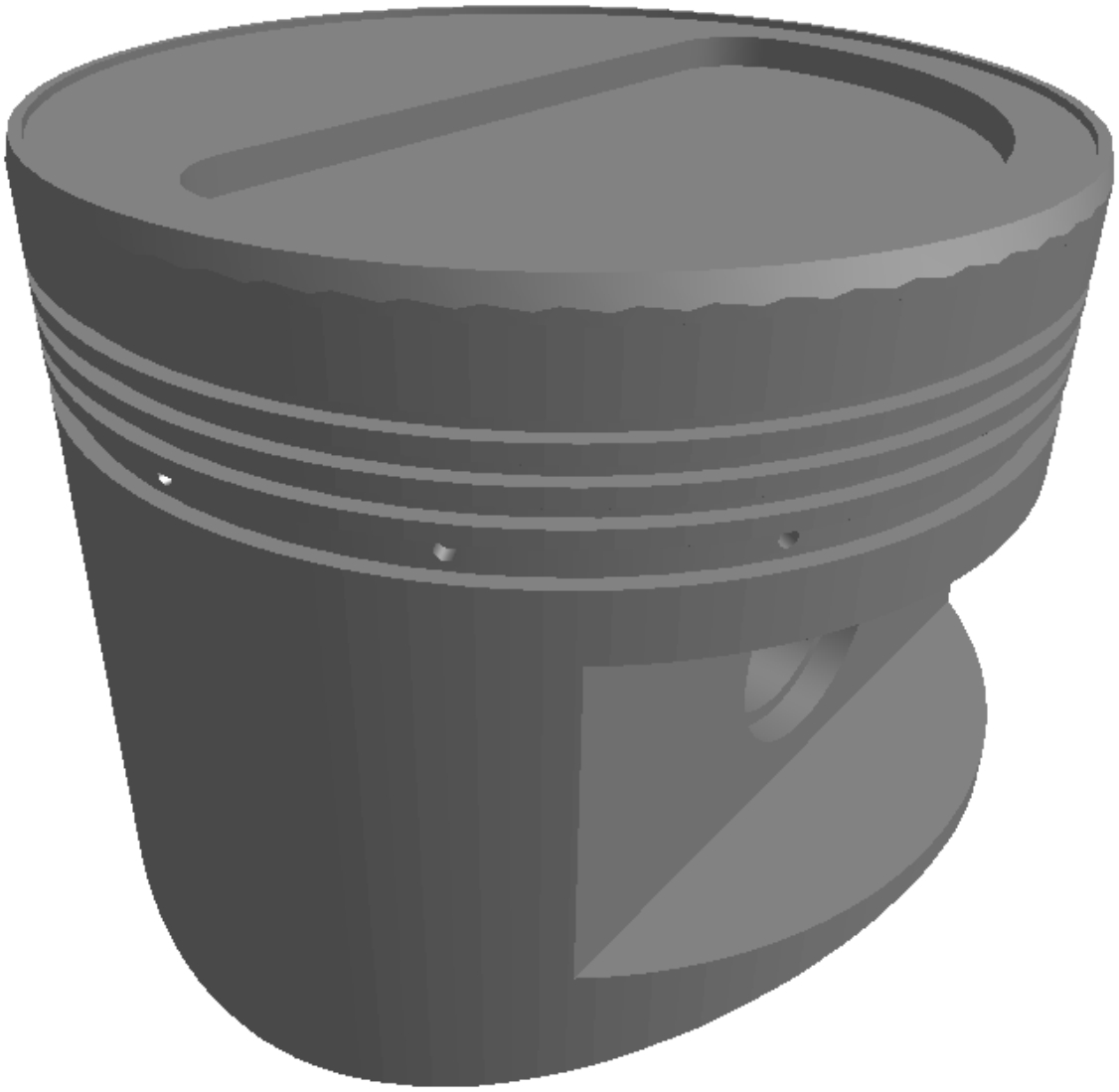

Figura 55. Modelo CSG de un pistón

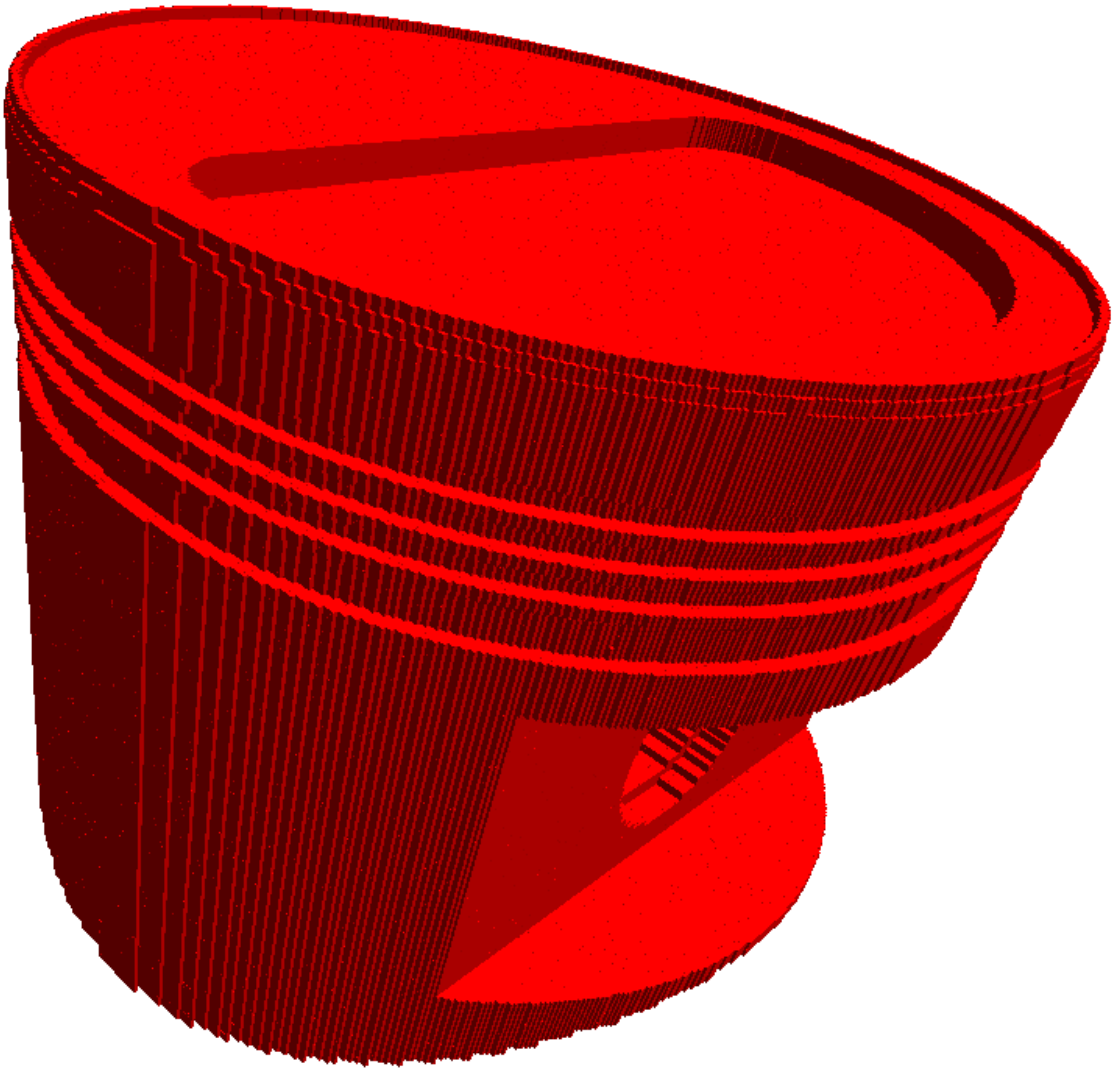

Figura 56. Voxelización del modelo CSG de un pistón

En la Figura 55 se muestra una un pistón de motor modelado únicamente mediante operaciones CSG. El modelo resulta de 35 operaciones sobre 33 primitivas, siendo la expresión RPN de la operación CSG la siguiente:

0 25 I 26 U 27 28 D 29 D 30 D 31 D 32 D 29 U 30 U D 1 D 2 D 3 D 4 D 5 D 6 D 7 D 8 D 9 D 10 D 11 D 12 D 13 D 14 D 15 D 16 D 17 D 18 D 19 D 20 D 2 1 D 22 D 23 D 24 D D

En la Figura 56 se muestra la voxelización del modelo CSG del modelo original mostrado en la Figura 55.

4.11.2 Modelo pieza CAD

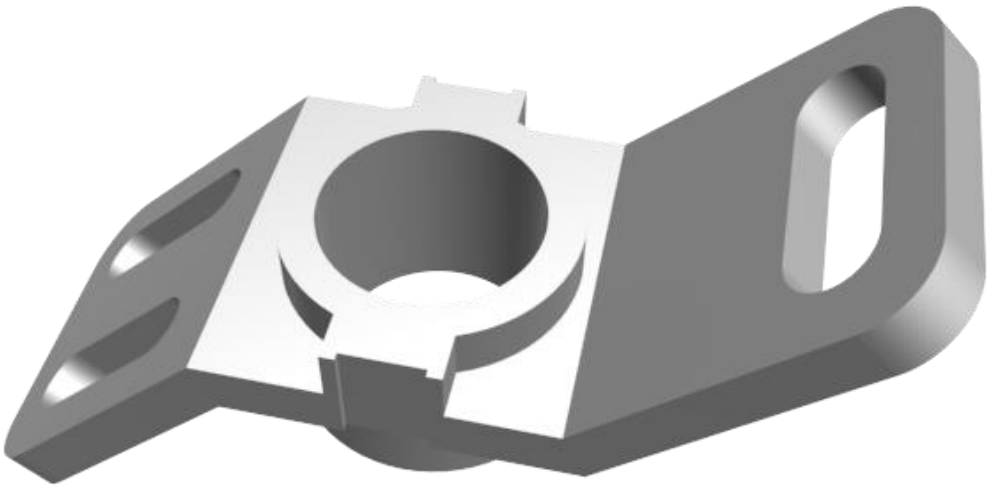

Figura 57. Modelo CSG de una pieza CAD

4.11.3 Modelo pirámide de Sierpinski

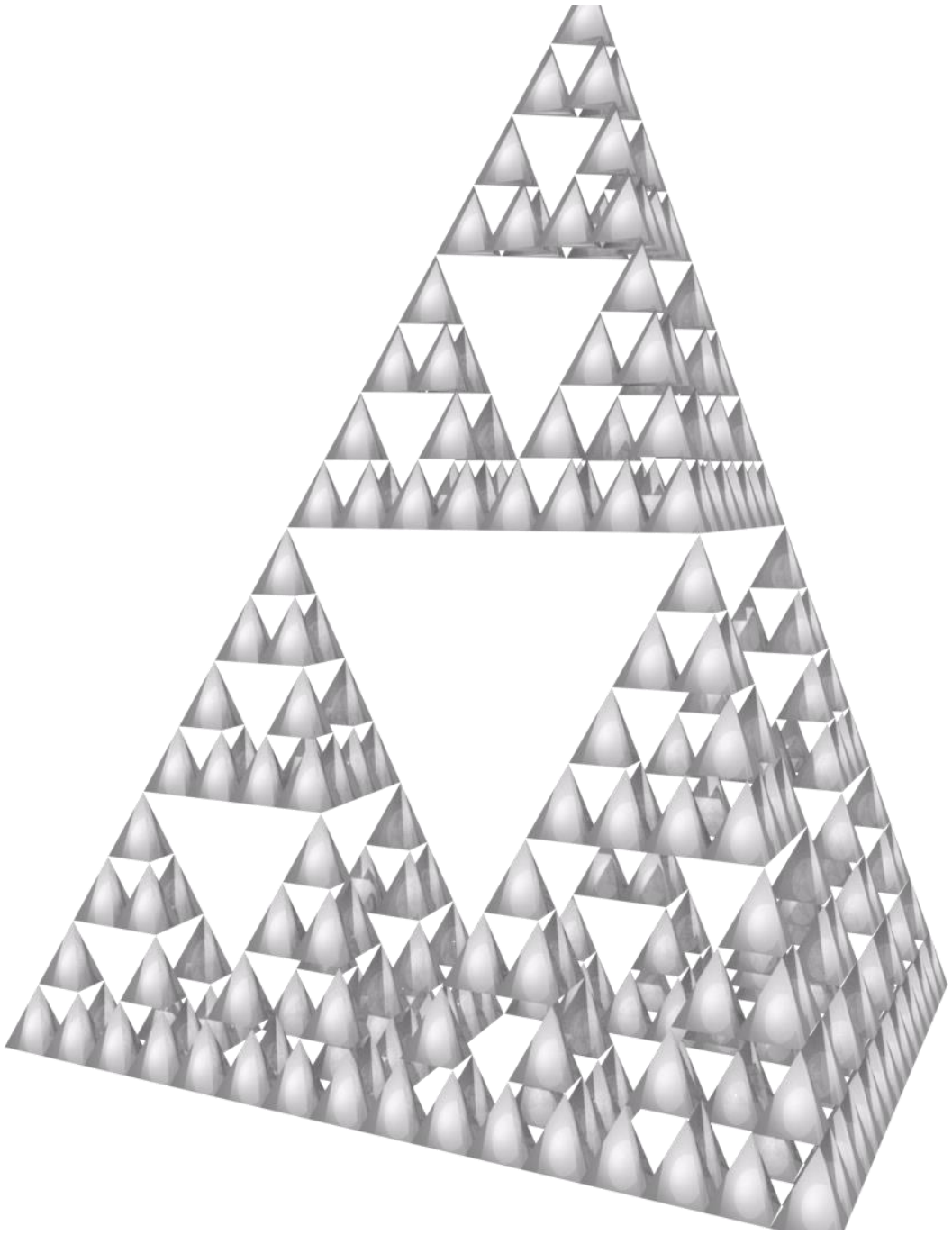

Figura 58. Modelo CSG de la pirámide de Sierpinski

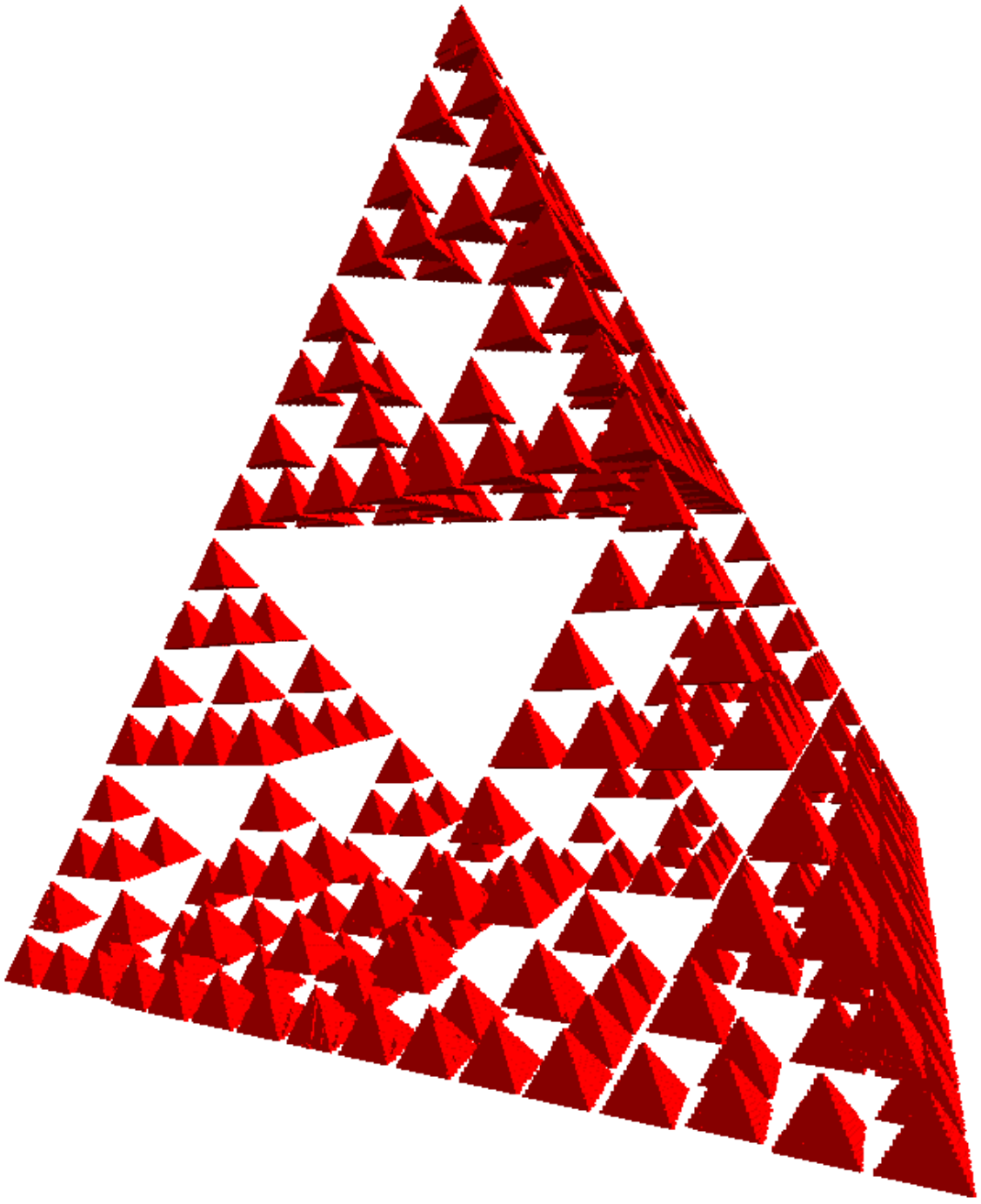

Figura 59. Voxelización del modelo CSG de la pirámide de Sierpinski

5 CONCLUSIONES Y PROPUESTAS FUTURAS

5.1 CONCLUSIONES

El paralelismo ha sido uno de los paradigmas en auge en la última década. En primer lugar fueron las CPUs las que trajeron al desarrollador la posibilidad de utilizar dicho paradigma. Y con la aparición de los primero pipelines programables, y posteriormente GPGPU, se incrementó el interés hacia dicho paradigma de desarrollo.

Sin embargo como ya hemos visto no fue hasta CUDA que no empezó a expandirse de forma potencial el interés por la programación paralela masiva. Dicha arquitectura está suponiendo la posibilidad de avanzar en las distintas investigaciones a pasos agigantados, enriqueciendo los campos de la biotecnología, la física, la inteligencia artificial…

Más aún, en el proyecto que me ocupa, ha supuesto una gran ventaja. Para el algoritmo estudiado se obtiene una ganancia de hasta 142. Todo esto teniendo en cuenta que el hardware gráfico con el cual se realiza la ejecución es una gama media de la anterior arquitectura a la actual.

Con la evolución del tiempo, las arquitecturas de los procesadores gráficos avanzarán enormemente, de tal modo que puede que incluso la CPU se convierta en un procesador de gestión y la GPU en un procesador puramente de procesamiento, de manera oficial.

Quedan por ver las ventajas de las nuevas arquitecturas por salir a consumidor, como es la Pascal de NVIDIA. No solo estas aumentando en potencia computacional, sino también en rendimiento energético, siendo esto último ideal para CPDs, así como para el IoT.

5.2 PROPUESTAS FUTURAS

El trabajo realizado en este proyecto es bastante extenso, aun así dado que este proyecto tiene una duración limitada, no ha sido posible plasmar todas las ideas que han ido surgiendo durante el desarrollo del mismo. A continuación enumero alguna de las ideas que más interesantes resultan:

- Implementar una versión del algoritmo utilizando divisiones espaciales específicas, aprovechando el paralelismo dinámico introducido con la arquitectura Maxwell.
- Realizar una extensión del proyecto de tal modo que se presente una versión de “Point in Solid” a partir del algoritmo aquí presentado.
- Realizar más aplicaciones del algoritmo, como puede ser simulación física de partículas, etc…

6 BIBLIOGRAFÍA

1. **Wikipedia.** Boundary representation. [Online] https://en.wikipedia.org/wiki/Boundary_representation.

2. —. *Ray casting.* [Online] https://en.wikipedia.org/wiki/Ray_casting.

3. *Inclusion test for free-form solids.* **Miras, J.Ruiz de and Feito, F.R.** Jaén, Spain : Elsevier, 1999, Computer & Graphics, Vol. 23, pp. 255-268.

4. **Wikipedia.** *Constructive solid geometry.* [Online] https://en.wikipedia.org/wiki/Constructive_solid_geometry.

5. **NVIDIA.** CUDA Occupancy Calculator. [Online] http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls.

6. **Jordan, Camille.** *Cours D'Analyse.* Paris : s.n., 1887. págs. 587-594. Vol. 3.

7. *Algorithm 112: Position of point relative to polygon.* **Shimrat, M.** [ed.] C. C. Gotlieb. 8, Alberta : ACM, August 8, 1962, Communications of the ACM, Vol. 5, p. 434. 0001-0782.

8. *Orientation, simplicity, and inclusion test for planar polygons.* **Feito, F., Torres, J. C. y Ureña, A.** 4, s.l. : Elsevier, 1995, Computer & Graphics, Vol. 19, págs. 595-600.

9. *Inclusion test for general polyhedra.* **Feito, F. y Torres, J.C.** 1, s.l. : Elsevier, 1997, Computer & Graphics, Vol. 21, págs. 23-30.

10. **DARPA.** THz Electronics. [En línea] http://www.darpa.mil/program/thz-electronics.

11. *Geometric algorithms on CUDA.* **Rueda, Antonio y Ortega, Lidia.** Jaén : Journal of Virtual Reality and Broadcasting, 2008, Vol. 200. 1860-2037.

12. ***TurboSquid.*** **[Online] http://www.turbosquid.com.**

13. Gahagan, Mark. GPUs: Doing More Than Just Games. *University of California, San Diego.* [Online] November 29, 2012. http://cseweb.ucsd.edu/classes/fa12/cse141/pdf/09/GPU_Gahagan_FA12.pdf.

14. Wikipedia. Tesla (microarchitecture). [Online] https://en.wikipedia.org/wiki/Tesla_(microarchitecture).

15. —. Fermi (microarchitecture). [Online] https://en.wikipedia.org/wiki/Fermi_(microarchitecture).

16. —. Kepler (microarchitecture). [Online] https://en.wikipedia.org/wiki/Kepler_(microarchitecture).

17. —. Maxwell (microarchitecture). [Online] https://en.wikipedia.org/wiki/Maxwell_(microarchitecture).

18. NVIDIA. NVIDIA GeForce GTX 680. [Online] http://www.nvidia.com/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf.

19. —. NVIDIA's Next Generation CUDATM Compute Architecture: Kepler TM GK110. [Online] http://www.nvidia.com/content/pdf/kepler/nvidia-kepler-gk110-architecture-whitepaper.pdf.

20. —. NVIDIA GeForce GTX 750 Ti. [Online] http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf.

21. —. Maxwell: The Most Advanced CUDA GPU Ever Made. [Online] September 18, 2014. http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/.

22. Wikipedia. CUDA. [En línea] https://en.wikipedia.org/wiki/CUDA.

23. —. Polytope. [Online] https://en.wikipedia.org/wiki/Polytope.

24. —. Simplex. [En línea] https://en.wikipedia.org/wiki/Simplex.

25. —. Barycentric coordinate system. [Online] https://en.wikipedia.org/wiki/Barycentric_coordinate_system.

26. Torres, Mario Salazar de. GPGPU y coprocesadores. *Prezi.* [En línea] https://prezi.com/thahqy3brb3g/gpgpu-y-coprocesadores/.

27. Intel. Xeon Phi Detail. [Online] http://www.intel.la/content/www/xl/es/processors/xeon/xeon-phi-detail.html.

28. Harris, Mark. Optimizing Parallel Reduction in CUDA. [Online] http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf.

7 ANEXOS

7.1 PRESUPUESTO

Para llevar a cabo el proyecto es necesario realizar una estimación del coste de los recursos utilizados durante el ciclo de desarrollo del mismo.

7.1.1 Recursos necesarios

En primer lugar, he de definir aquellos recursos que vamos a utilizar durante todo el proyecto.

Aunque en este caso el proyecto lo vaya a desarrollar yo únicamente con la ayuda de mi tutor, si el proyecto se desarrollase en un ámbito empresarial, quizás se haría uso de hasta 2 programadores con categoría Junior, como es el caso en la planificación presentada.

En la Tabla 44 se muestra la lista de los recursos necesarios, así como el coste de los mismos.

Recurso	Cantidad	Importe
Analista funcional	1	15,63 €/hora
Programador Senior	1	14,58 €/hora
Programador Junior	2	10,94 €/hora
Ordenador fijo	1	736 €
Tarjeta gráfica GTX 660Ti	1	220 €

Tabla 44. Detalles de recursos en términos presupuestarios.

Al respecto de los recursos cabe mencionar lo siguiente:

- El coste por hora dado para el personal humano es resultado de un estudio de las ofertas laborales en el mercado actual.
- En este caso el analista funcional sería el tutor del proyecto, cuyo objetivo en el mismo es el de revisarlo y asistir en la resolución de problemas que se presenten durante el desarrollo del mismo.
- Respecto al programador Senior, dicho rol sería desempleado por el alumno encargado de la realización del proyecto.
- Los programadores Junior se encargarían de realizar tareas que el programador Senior no podría realizar debido a que de lo contrario sobrecargaría su jornada laboral, aunque dado que en este caso, únicamente estaría durante el proceso de desarrollo el alumno, esté deberá asumir los roles tanto de Junior, como Senior, al mismo tiempo.

- El ordenador utilizado durante el proceso de desarrollo se trata de un ordenador personal de gama media con un procesador Intel i5-2500k, 8GB de RAM, 2TB de HDD, y una placa base Asus de gama media-baja.
- La tarjeta gráfica GTX 660 Ti es la pieza de tecnología más importante del proyecto, pues es sobre esta sobre la cual se van a realizar los cálculos estadísticos del algoritmo optimizado.

7.1.2 Desglose por tareas

Como ya se comentó en el apartado de planificación del proyecto, para efectos prácticos, se ha establecido la jornada laboral de 8 horas.

En la Tabla 45 se muestra una lista donde aparece el nombre, duración en horas, así como el coste de las tareas desarrolladas por el tutor del proyecto, el cual desempeña el rol de analista funcional durante el mismo.

Tarea	Duración (h)	Importe
Reunión con el tutor (1)	2 horas	31,26 €
Reunión con el tutor (2)	2 horas	31,26 €
Reunión con el tutor (3)	2 horas	31,26 €
Reunión con el tutor (4)	2 horas	31,26 €
Reunión con el tutor (5)	2 horas	31,26 €
Reunión con el tutor (6)	2 horas	31,26 €
Reunión con el tutor (7)	2 horas	31,26 €
Reunión con el tutor (8)	2 horas	31,26 €
Reunión con el tutor (9)	2 horas	31,26 €
Reunión con el tutor (10)	2 horas	31,26 €
Reunión con el tutor (11)	2 horas	31,26 €
Reunión con el tutor (12)	2 horas	31,26 €
Revisión final del proyecto	72 horas	1.125,36 €

Tabla 45. Detalle del coste de las tareas desarrolladas por el tutor

En la Tabla 46 se muestra una lista donde aparece el nombre, duración en horas, así como el coste de las tareas desarrolladas por el alumno, el cual desempeña el rol de programador Senior durante el mismo.

Tarea	Duración (h)	Importe
Reunión con el tutor (1)	2 horas	29,16 €
Estudio y evaluación del proyecto	24 horas	349,92 €
Búsqueda bibliográfica	48 horas	699,84 €
Reunión con el tutor (2)	2 horas	29,16 €
Revisión bibliográfica	32 horas	466,56 €
Inicio de la documentación	40 horas	583,20 €
Reunión con el tutor (3)	2 horas	29,16 €
Configuración del toolkit de CUDA	8 horas	116,64 €
Elaboración de la versión 0 del algoritmo	24 horas	349,92 €
Implementación CUDA de la versión 0 del algoritmo	40 horas	583,20 €
Reunión con el tutor (4)	2 horas	29,16 €
Obtención parcial de estadísticas del algoritmo 0	24 horas	349,92 €
Revisión de la versión 0 del algoritmo	48 horas	699,84 €
Reunión con el tutor (5)	2 horas	29,16 €
Elaboración de la versión definitiva del algoritmo	24 horas	349,92 €
Implementación CUDA de la versión definitiva	48 horas	699,84 €
Reunión con el tutor (6)	2 horas	29,16 €
Obtención parcial de estadísticas del algoritmo definitivo	24 horas	349,92 €
Revisión de la versión definitiva del algoritmo	48 horas	699,84 €
Reunión con el tutor (7)	2 horas	29,16 €
Desarrollo de una aplicación práctica para el algoritmo	56 horas	816,48 €
Obtención de los datos necesarios para la ejecución de la aplicación práctica del algoritmo	16 horas	233,28 €
Reunión con el tutor (8)	2 horas	29,16 €
Obtención de las estadísticas para la versión monohilo del algoritmo	72 horas	1.049,76 €
Reunión con el tutor (9)	2 horas	29,16 €
Obtención de las estadísticas para la versión multi-hilo del algoritmo	72 horas	1.049,76 €
Reunión con el tutor (10)	2 horas	29,16 €
Obtención de las estadísticas para la versión CUDA del algoritmo	48 horas	699,84 €
Generación de las gráficas de las estadísticas	24 horas	349,92 €
Reunión con el tutor (11)	2 horas	29,16 €
Terminar la memoria del proyecto	64 horas	933,12 €

Tabla 46. Detalle del coste de las tareas desarrolladas por el alumno

En la Tabla 47 se muestra una lista donde aparece el nombre, duración en horas, así como el coste de las tareas desarrolladas por el primer desarrollador Junior.

Tarea	Duración (h)	Importe
Implementación monohilo de la versión 0 del algoritmo	32 horas	350,08 €
Implementación multi-hilo de la versión definitiva	40 horas	437,60 €
Completar la memoria del proyecto (2)	72 horas	787,68 €

Tabla 47. Detalle del coste de las tareas desarrolladas por el programador Junior 1

En la Tabla 48 se muestra una lista donde aparece el nombre, duración en horas, así como el coste de las tareas desarrolladas por el segundo desarrollador Junior.

Tarea	Duración (h)	Importe
Implementación monohilo de la versión definitiva	32 horas	350,08 €
Completar la memoria del proyecto (1)	72 horas	787,68 €
Búsqueda de los modelos de prueba	8 horas	87,52 €

Tabla 48. Detalle del coste de las tareas desarrolladas por el programador Junior 2

En la Tabla 49 se muestra una lista donde aparece el rol, trabajo en horas y costo durante todo el proyecto para cada componente del personal necesario para el desarrollo del mismo. Esto se muestra a modo de resumen.

Tarea	Trabajo (h)	Coste
Analista funcional	96 horas	1.500,48 €
Programador Senior	890 horas	12.976,20 €
Programador Junior 1	144 horas	1.575,36 €
Programador Junior 2	112 horas	1.225,28 €

Tabla 49. Resumen del coste del personal del proyecto

7.1.3 Resumen final

En total se invierten unas **1242 horas** en el proyecto y supone un coste de personal de unos **17.277,32 €**

En la Tabla 50 se muestra el un resumen del coste de cada uno de los recursos del proyecto, resultando en un coste total de **18.233,32 €**

Concepto	Coste
Costes de desarrollo	17.277,32 €
Ordenador fijo	736 €
Programador Junior 1	220 €
TOTAL	18.233,32 €

Tabla 50. Coste de los recursos del proyecto

7.2 MANUAL DE USUARIO DE LA SOLUCIÓN

En este apartado se explica el funcionamiento de las distintas aplicaciones de la solución entregada en el código fuente.

Es importante notar que todos los proyectos adjuntos a la solución pueden ser compilados tanto para 32 bits como para 64 bits.

Para la ejecución de las aplicaciones se requerirá de la versión de CUDA 6.5 instalada, así como del "Visual C++ Redistributable Packages for Visual Studio 2013" instalado, bien la versión de x86, x64.

Todas las aplicaciones de la solución se encuentran en su versión "Release" tanto en 32 bits como en 64 bits, dentro de la carpeta "Builds" de la raíz del proyecto.

Cabe mencionar además que se ha creado una nueva representación llamada ITD, apta para el nuevo algoritmo presentado en el proyecto. Más adelante se presentará una utilidad que permite convertir una representación B-Rep a la representación ITD.

Todas las aplicaciones que implementan el algoritmo con el objetivo de obtener estadísticas, evalúan los modelos sobre un cubo de puntos distribuidos uniformemente a lo largo del AABB de los mismos. Las dimensiones de dicho cubo se presentan en los argumentos de las aplicaciones como "Width", "Height" y "Deep", correspondiendo al número de puntos en cada una de las 3 dimensiones del cubo.

Las versiones "ST_Grid", "MT_Grid" y "CUDA_Grid" constan de una interfaz de visualización de resultado. El movimiento de la cámara se controla con el ratón, y además disponen de varias asignaciones en el teclado para controlar distintos aspectos. Tales como:

- **"A".** Se utiliza para mostrar/ocultar los ejes de coordenadas.
- **"C".** Cambia el modo de renderizado a nube de puntos.
- **"W".** Cambia el modo de renderizado a alambre.
- **"F".** Cambia el modo de renderizado a relleno de polígonos.
- **"S".** Cambia entre el sistema de coordenadas global/local.
- **"T".** Muestra/oculta la nube de puntos que se encuentran dentro del modelo evaluado.
- **"M".** Muestra/oculta el modelo evaluado.

7.2.1 ST

Esta aplicación incluye la implementación “STP” del algoritmo. Se trata de una aplicación de línea de comandos, cuya salida consiste en:

<Número de nodos evaluados>, <Número de puntos evaluados>, <Tiempo total transcurrido durante la ejecución>, <Media del tiempo de ejecución de un punto individual>, <Desviación estándar del tiempo de ejecución de un punto individual>

La sintaxis de la aplicación es la siguiente:

ST_Win<32|64>.exe <Width> <Height> <Deep> <File.itd>

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa.
- **File.itd**. La representación ITD del modelo evaluado.

7.2.2 ST_Grid

Esta aplicación incluye la implementación “STG” del algoritmo. Se trata de una aplicación de línea de comandos, cuya salida consiste en:

<Número de nodos evaluados>, <Número de puntos evaluados>, <Tiempo total transcurrido durante la ejecución>

La sintaxis de la aplicación es la siguiente:

ST_Grid_Win<32|64>.exe <Width> <Height> <Deep> [File.obj]

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa.
- **File.obj**. La representación B-Rep del modelo evaluado con formato OBJ. Es importante tener en cuenta que debe existir en la misma ruta del fichero OBJ el fichero “File.obj.itd”, sino, no se realizará la evaluación contra el algoritmo.

NOTA. Si se especifica el parámetro File.obj la aplicación tratará de hacer la evaluación contra el algoritmo y después se cerrará, de lo contrario se abrirá un dialogo modal para buscar el fichero y la aplicación mostrará visualmente tanto la representación B-Rep, como los puntos evaluados dentro del modelo.

7.2.3 MT

Esta aplicación incluye la implementación "MTP" del algoritmo. Se trata de una aplicación de línea de comandos, cuya salida consiste en:

<Número de nodos evaluados>, <Número de puntos evaluados>, <Tiempo total transcurrido durante la ejecución>, <Media del tiempo de ejecución de un punto individual>, <Desviación estándar del tiempo de ejecución de un punto individual>

La sintaxis de la aplicación es la siguiente:

MT_Win<32|64>.exe <Width> <Height> <Deep> <Threads> <File.itd>

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa.
- **Threads**. El número de hilos con el que ejecutar la evaluación
- **File.itd**. La representación ITD del modelo evaluado.

7.2.4 MT_Grid

Esta aplicación incluye la implementación "MTG" del algoritmo. Se trata de una aplicación de línea de comandos, cuya salida consiste en:

<Número de puntos evaluados>, <Número de puntos evaluados>, <Tiempo total transcurrido durante la ejecución>, <Máximo tiempo total transcurrido dentro de los hilos lanzados>

La sintaxis de la aplicación es la siguiente:

```
MT_Grid_Win<32|64>.exe <Width> <Height> <Deep> [Threads] [File.obj]
```

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa.
- **Threads**. El número de hilos lanzados para la evaluación del modelo contra el algoritmo. Si no se especifica el valor, se lanzarán tantos hilos como núcleo tenga el nodo de procesamiento.
- **File.obj**. La representación B-Rep del modelo evaluado con formato OBJ. Es importante tener en cuenta que debe existir en la misma ruta del fichero OBJ el fichero "File.obj.itd", sino, no se realizará la evaluación contra el algoritmo.

NOTA. Si se especifica el parámetro File.obj la aplicación tratará de hacer la evaluación contra el algoritmo y después se cerrará, de lo contrario se abrirá un dialogo modal para buscar el fichero y la aplicación mostrará visualmente tanto la representación B-Rep, como los puntos evaluados dentro del modelo.

7.2.5 CUDA_Reduction

Esta aplicación incluye la implementación "R660" del algoritmo. Se trata de una aplicación de línea de comandos, cuya salida consiste en:

```
<Número de nodos evaluados>, <Número de puntos evaluados>, <Tiempo total transcurrido durante la ejecución>, <Tiempo interno de ejecución>, <Tiempo invertido en copiar la memoria constante>, <Tiempo invertido en copiar los nodos>, <Tiempo invertido en copiar los puntos >, <Tiempo invertido en copiar los resultados de las evaluaciones>
```

La sintaxis de la aplicación es la siguiente:

```
CUDA_Reduction<32|64>.exe <Width> <Height> <Deep> <File.itd>
```

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa.
- **File.itd**. La representación ITD del modelo evaluado.

7.2.6 CUDA_Grid

Esta aplicación incluye la implementación “G660” del algoritmo. Se trata de una aplicación de línea de comandos, cuya salida consiste en:

```
<Número de nodos evaluados>, <Número de puntos evaluados>, <Tiempo total transcurrido durante la ejecución>, <Tiempo interno de ejecución>, <Tiempo invertido en copiar la memoria constante>, <Tiempo invertido en copiar los nodos>, <Tiempo invertido en copiar los puntos >, <Tiempo invertido en copiar los resultados de las evaluaciones>
```

La sintaxis de la aplicación es la siguiente:

```
CUDA_Grid<32|64>.exe [Width] [Height] [Deep] [Parts] [File.obj]
```

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa. El valor por defecto es 256.
- **Parts**. La ejecución se dividirá en (1 << Parts) partes. El valor por defecto es 5.
- **File.obj**. La representación B-Rep del modelo evaluado con formato OBJ. Es importante tener en cuenta que debe existir en la misma ruta del fichero OBJ el fichero “File.obj.itd”, sino, no se realizará la evaluación contra el algoritmo.

NOTA. Si se especifica el parámetro File.obj la aplicación tratará de hacer la evaluación contra el algoritmo y después se cerrará, de lo contrario se abrirá un dialogo modal para buscar el fichero y la aplicación mostrará visualmente tanto la representación B-Rep, como los puntos evaluados dentro del modelo.

7.2.7 CUDA_CSG

Esta se trata de la aplicación práctica del algoritmo, la cual dada una expresión RPN, y una serie de primitivas ITD construye un modelo CSG.

La sintaxis de la aplicación es la siguiente:

```
CUDA_Grid<32|64>.exe <RPNExpression> [<Width> <Height> <Deep>] [“File[0]; File[1]; File[2]; … File[n]”] [Parts]
```

La descripción de los parámetros es la siguiente:

- **Width, Height, Deep**. Las dimensiones del cubo sobre el cual se evalúa. El valor por defecto es 256. Es importante tener en cuenta que si se especifica solo se tendrán en cuenta si se especifican los 3 parámetros.
- **Parts**. La ejecución se dividirá en (1 << Parts) partes. El valor por defecto es 5.
- **“File[0]; File[1]; File[2] … File[n]”**. Corresponde con la lista de primitivas ITD utilizadas al evaluar la expresión CSG. Esta se encuentra separada por “;” y encerrada en dobles comillas. Si no se seleccionan las suficientes primitivas la aplicación mostrará un fallo y terminará la ejecución.

NOTA. Si no se especifican la lista de ficheros correspondientes a las primitivas, aparecerá un dialogo para seleccionarlos.

7.2.8 OBJ2ITD

Esta es la aplicación que convierte las representaciones B-Rep con el formato OBJ a la representación ITD necesaria para la ejecución del algoritmo presentado en el proyecto.

Esta aplicación no acepta parámetros por consola, sin embargo al abrirla muestra un cuadro de dialogo en el que se te pide que selecciones un fichero de formato OBJ, el cual convertirá a ITD en la misma ruta del fichero original.