Implementación en CUDA de un algoritmo de flujo óptico denso

Proyecto de fin de carrera.
Facultad de informática.
Universidad de Las Palmas de Gran Canaria.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Leonardo Lanchas Rolando
Las Palmas de Gran Canaria
07 - 2017

Proyecto fin de carrera de la Facultad de Informática de la Universidad de Las Palmas de Gran Canaria presentado por el alumno:

Leonardo Lanchas Rolando

Título del Proyecto: Implementación en CUDA de un algoritmo de flujo óptico denso

Tutor: Carmelo Cuenca Hernández
Tutora: Francisca Quintana Domínguez

A mi familia y amigos.

Agradecimientos

Me gustaría agradecer en primer lugar a mis tutores Carmelo Cuenca Hernández y Francisca Quintana Domínguez toda la confianza depositada en mí desde el comienzo de este proyecto fin de carrera, tanto a nivel personal como académico.

Me gustaría agradecer a todo el Departamento de Informática y Sistemas de la Universidad de Las Palmas de Gran Canaria haberme formado académicamente durante los años de carrera.

El día a día en nuestra carrera, las largas prácticas, las horas de estudio, los momentos más duros y también los más agradables, durante más de cinco años, no habrían sido lo mismo sin Crístofer, Borja y Jorge, que comenzaron siendo mis compañeros y se han convertido en grandes amigos. Por su paciencia, su ayuda, su apoyo y su amistad, muchas gracias.

Sin duda he podido llegar hasta aquí y soy como soy gracias a las dos personas que me han criado, cuidado, enseñado, ayudado, apoyado y querido siempre, mis padres, Roberto y Gloria. A ellos les debo todo y jamás podré agradecerles lo mucho que han hecho por mí cada día en todos los aspectos, ni expresar lo mucho que les quiero. Por eso y por todo lo demás, muchísimas gracias a los dos por estar ahí en todo momento y por hacer por mí lo imposible.

Una persona de la que aprendo cada día, a la que he admirado siempre, a la que quiero y que, sobre todo, me soporta, es mi hermana Erika. Ella me ha enseñado a ser fiel a uno mismo y que lo más importante es ser buena persona. Para mi hermana también es este proyecto y espero que disfrute cada día hasta conseguir todas sus metas. No cambies nunca.

Antes de llegar hasta aquí, tuve la oportunidad y la gran suerte de vivir una de las etapas más bonitas de mi vida junto a mis amigos de siempre. Aunque a veces no disponga de mucho tiempo, siempre les tengo presente. A Esaú, Benjamín, Romina, Micaela y a todos y cada uno de los demás, gracias por lo que hemos vivido juntos y por quererme como soy

Agradezco también al resto de mis familiares el apoyo y el cariño que he recibido a lo largo de mi vida. De cada uno he aprendido muchísimo en todos los sentidos y les querré y recordaré toda la vida.

A familiares, amigos y profesores, muchas gracias.

Leonardo

Índice

Listado de ilustraciones

Introducción

El flujo óptico se define como el desplazamiento existente entre los píxeles de imágenes bidimensionales que han sido tomadas por una sola cámara en distintos instantes de tiempo. En principio, se desconoce a qué se debe dicho desplazamiento. Las imágenes no son más que las proyecciones del movimiento de los elementos en escenas tridimensionales por lo que con sólo una proyección no es posible determinar de forma unívoca la profundidad de los puntos. Por ese motivo, no siempre el movimiento descrito en el flujo óptico se corresponde con el de la escena real.

El cálculo del flujo óptico es uno de los problemas fundamentales en el campo de la visión por computador. Sirve como base para una amplia gama de aplicaciones, tales como: detección de movimiento, seguimiento de objetos y personas, reconstrucción tridimensional de escenas, detección de colisiones, análisis de imágenes médicas, de seguridad y vigilancia, procesamiento de imágenes de satélite, etc.

Con este proyecto vamos a explorar algunas ideas importantes, tales como el estudio de los métodos que explotan la información redundante entre cuadros que no son consecutivos con el fin de obtener soluciones más robustas y precisas al mismo tiempo que la aplicación de estrategias avanzadas para acelerar los esquemas numéricos que se generan. Dentro de estas estrategias, se considera el uso de hardware especializado para la aplicación de nuestros métodos, tales como unidades de procesamiento gráfico (GPU), el uso de las bibliotecas para la paralelización de algoritmos numéricos o realizaciones más avanzadas.

Existen diversos enfoques de entre los cuales, los métodos variacionales destacan por la precisión de sus soluciones. Dichos métodos consisten en una técnica basada en la minimización de energías en la que la solución debe cumplir una serie de restricciones impuestas en el modelo. A continuación se enumeran algunas de sus ventajas:

- Campos de desplazamientos densos: se dispone de un valor de desplazamiento para todos los píxeles de la imagen de forma que si la

información local no es suficiente para estimar la solución se utiliza la información de los vecinos. Por su parte, otro tipo de técnicas como por ejemplo las basadas en correlación, calculan la solución en determinados puntos de la imagen y la única forma de obtener soluciones densas es a través de un proceso de interpolación.

- Restricciones reflejadas en el modelo de energía: Todas las restricciones que se imponen en el método están reflejadas en el modelo de energía. No existe ninguna restricción que se implemente y que no aparezca en dicho modelo.

- Base matemática sólida: El modelado de la energía y el proceso de minimización se sustenta sobre el hecho que asegura la existencia y unicidad de la solución, así como la convergencia del método si se cumple una serie de condiciones.

Los modelos de energía definidos en este proyecto se componen de dos partes: (1) el término de ligadura y (2) el término de suavizado. El término de ligadura asume que cierta propiedad en la imagen no varía a lo largo del tiempo; mientras que el de suavizado supone cierta restricción de suavidad en el flujo. La minimización del modelo de energía genera un sistema de ecuaciones diferenciales en derivadas parciales. Para la resolución de este sistema de ecuaciones se suele recurrir a un método de descenso por gradiente. Este tiene la ventaja de ser fácil de implementar y la convergencia es relativamente rápida. Existen otras técnicas como SOR que convergen con un número menor de iteraciones.

Objetivos

La estimación del flujo óptico ha sido objeto de estudio durante mucho tiempo. A pesar de la amplia y profunda investigación que ha sido realizada, hoy en día se siguen haciendo numerosas aportaciones que han conseguido incrementar su precisión y robustez. Al principio, los métodos utilizaban sólo dos imágenes. Con el objetivo de aumentar la precisión de los resultados pasó a emplearse la información provista por los frames vecinos surgiendo de esta forma los primeros métodos espacio-temporales que han supuesto un gran avance.

El objetivo principal de este proyecto es investigar si a través de la paralelización de los procedimientos gracias a los avances tecnológicos en cuanto a hardware se refiere, se consigue mejorar su exactitud y rapidez. Para ello, se desarrollará una aplicación escrita en C++ con la que se realizarán experimentos diversos y se analizarán los resultados producidos para verificar si es útil avanzar por este camino o no.

Las tareas que se deben abordar para realizar dicha aplicación son las siguientes:

- Entender el algoritmo que ha sido desarrollado previamente
- Diseñar una interfaz gráfica fácil de usar que permita manejar la aplicación de manera intuitiva de forma que su curva de aprendizaje se ínfimo.
- Aprender a programar con CUDA
- Diseñar la arquitectura de la aplicación
- Paralelizar el algoritmo propuesto mediante CUDA
- Realizar tests en un PC de sobremesa

Estructura del documento

Con el fin de facilitar la lectura del documento se presenta en este apartado su estructura general y aspectos más relevantes.

Tras haber realizado la introducción del proyecto y haber marcado los principales objetivos del mismo, se exponen a continuación cada una de sus fases. En primer lugar se presentan los estudios realizados acerca de la situación actual (estado del arte) y las herramientas existentes antes de comenzar. Posteriormente, se detallan la metodología seguida a lo largo del ciclo de vida, además de los recursos necesarios para su elaboración.

En los siguientes apartados se detallarán el análisis, diseño y desarrollo de forma exhaustiva para finalizar el documento con los resultados y conclusiones obtenidas, junto con el posible trabajo futuro que podría realizarse. Además, se indica la bibliografía consultada y se presentan los anexos del documento con manuales de uso e instalación de la aplicación.

En lo que respecta a la elaboración, se ha centrado principalmente en la implementación del algoritmo para que pueda ser ejecutado en la GPU a la par que en la creación de una interfaz gráfica lo más sencilla e intuitiva posible con el objetivo de facilitar la comprensión de los resultados al usuario.

En cada una de las fases presentadas en este documento se podrán encontrar diferentes apartados, organizados de manera que se facilite la lectura y compresión de la misma. A lo largo de la memoria se hará referencia a la bibliografía consultada utilizando el formato indicado por la IEEE en 2006 que consiste en una referencia numérica a la bibliografía entre corchetes, por ejemplo: [1] hace referencia al primer libro citado en el apartado de la bibliografía.

En el "Anexo 1 - Manual de Usuario" podrán observarse imágenes y explicaciones detalladas de cada una de los aspectos desarrollados. El código fuente y el resultado completo de la aplicación que ha sido implementada se podrán consultar en el CD adjunto a esta memoria.

Situación actual

En este apartado se presenta un estudio del estado del hardware tanto general como específico disponible además del software, útil para la tarea que nos ocupa así como las tendencias que se están siguiendo en nuevos desarrollos.

Además, se describe la situación actual del problema de la estimación del flujo. Se hace mención a los métodos más relevantes describiendo algunas de las técnicas más innovadoras surgidas en los últimos años que han servido de referencia a otros métodos desarrollados con posterioridad.

Estado del hardware

Se ha realizado un estudio de la situación presente de diversas herramientas que pueden ser útiles para llevar a cabo nuestro propósito. Se ha observado una gran variedad de formatos y estructuras por lo que este apartado se centrará en presentar un resumen de los diferentes equipos que los fabricantes ponen a disposición del público general.

Antes de pasar a enumerar de manera exhaustiva cada una de las características del hardware útil actualmente, es necesario tener en mente la conocida Ley de Moore.

La ley de Moore expresa que aproximadamente cada dos años se duplica el número de transistores en un circuito integrado. [1]

Se trata de una ley empírica, formulada por el cofundador de Intel, Gordon E. Moore, el 19 de abril de 1965, cuyo cumplimiento se ha podido constatar hasta hoy.

Más tarde, en 1975, modificó su propia ley al corroborar que el ritmo bajaría, y que la capacidad de integración se duplicaría aproximadamente cada 18 meses. [2, pp. 11-13]

Sin embargo, el propio Moore determinó una fecha de caducidad para su ley: "Mi ley dejará de cumplirse dentro de 10 o 15 años -desde 2007". [3]

La consecuencia directa de la ley de Moore es que los precios bajan al mismo tiempo que las prestaciones suben: la computadora que hoy vale 3000 dólares costará la mitad al año siguiente y estará obsoleta en dos años. En 26 años el número de transistores en un chip se ha incrementado 3200 veces.

Vanguardia

Supercomputadores

Hoy en día la supercomputadora con mayor capacidad de cómputo es Sunway TaihuLight, que presenta una velocidad de procesamiento de 93 Petaflops [4].

No obstante, hace apenas 5 años, el que ocupaba la cabeza de la lista era el Titán capaz de procesar a 17,93 Petaflops [5]

Con esto se pretende poner de relieve la extremada rapidez con la que se producen los cambios en el hardware, reafirmando así lo establecido por Moore en 1965.

Es preciso mencionar brevemente al IBM Roadrunner ya que tras menos de 5 años de uso, será puesto fuera de servicio tras haber costado 120 millones de dólares en construirse y consumir 1.200.000 de dólares al año en energía. Hoy Titán, la número 1 es 17 veces más rápida y consume el 25% de esa energía. [6]

Raspberry PI

Raspberry Pi es una placa computadora (Single Board Computer o SBC) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

Incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos

modos "Turbo" que permiten hacer overclock hasta 1 GHz) [7], un procesador gráfico (GPU) VideoCore IV, y 512 MB de memoria RAM. No presenta disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa.

El modelo B cuesta $35 y el A $25.

Por otra parte, no viene con reloj en tiempo real, por lo que el sistema operativo debe usar un servidor de hora en red, o pedir al usuario la hora en el momento de arrancar el ordenador. Sin embargo se podría añadir un reloj en tiempo real (como el DS1307) con una batería mediante el uso de la interface I^2C. [7]

Por si sola esta placa no sirve para los menesteres de este proyecto. No obstante, resulta interesante lo que se puede lograr cuando se juntan muchas de ellas.

Ingenieros del Computational Engineering and Design Research Group de la Facultad de Ingeniería de la Universidad de Southampton, en Reino Unido, liderados por Simon Cox, han juntado 64 Raspberry Pi y han montado una especie de supercomputador de bajo coste con el que pueden realizar experimentos varios a un precio asumible, 3000 € (sin contar cableado ni switches).

Los racks, con sus respectivos nodos independientes, fueron creados con una estructura de Lego perfectamente ampliable. [8]

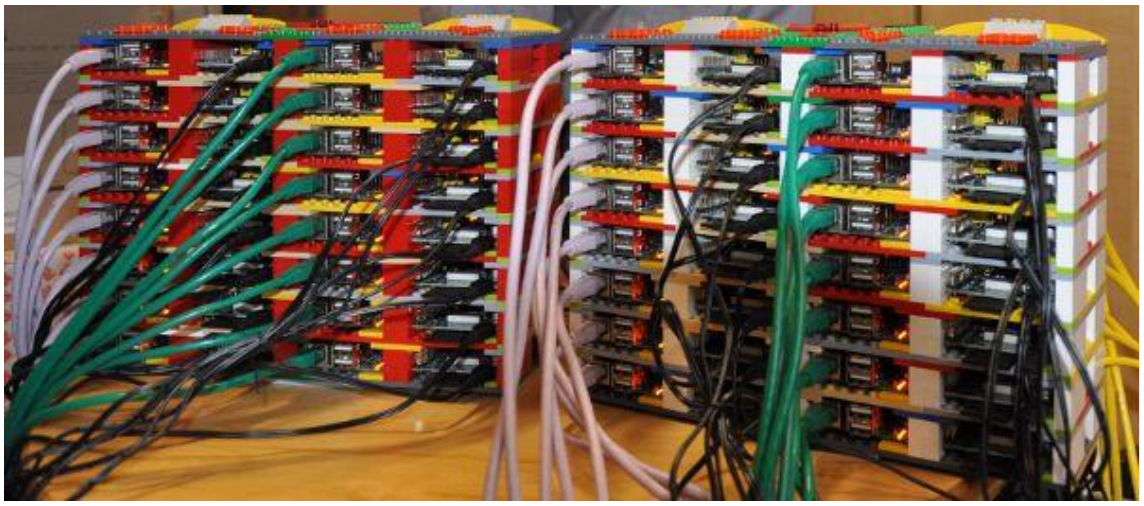


Imagen 1 – Supercomputador low-cost

[9]

Evidentemente no resulta la herramienta de cálculo más potente, pero sí uno de las más pequeñas, de las que menos consumen y también de las que menos calor generan.

El atractivo de este invento radica en el bajo presupuesto para tener un pequeño ordenador de "altas prestaciones". Si bien el resultado dista enormemente de los actuales, debería ser más que suficiente para ciertas labores.

Algunos datos generales son los siguientes:

- 64 núcleos ARM a 700 MHz. (aunque puede subirse hasta 1 GHz.)
- 1 TB de espacio de almacenamiento total basado en tarjetas SD de 16 GB
- 32 GB de memoria RAM (a razón de 512 MB por cada SoC)

Potencia total de unos 200 vatios (a razón de unos 3 vatios por cada placa, si bien este valor depende enormemente del porcentaje de uso de la CPU así como de los periféricos conectados). Equivale a un ordenador doméstico de gama media-alta.

Supercomputador Atlante

Como ejemplo local tenemos el situado en Canarias. Es un cluster de memoria distribuida, consistente en 84 Blades IBM JS21, cada uno con dos

procesadores dual-core PowerPC 970 MP, 8GB de memoria, y un adaptador PCI-X Myrinet 2000 para aplicaciones MPI de gran ancho de banda y baja latencia. Además de los discos locales de scratch, estos nodos de computación se encuentran conectados vía gigabit Ethernet a un sistema de ficheros GPFS de 3TBytes, alojado en 2 sistemas IBM p510 Power5.

Se trata de un sistema con un total de 336 núcleos de procesamiento, una potencia de cálculo total de 3.30 Tflops y 3Tbytes de almacenamiento, funcionando bajo el sistema operativo SUSE Linux 10.0 [10]

Información de los nodos		
Computación (82 nodos JS21 Bladeserver)	Usuario (2 JS21 Bladeserver)	Gestión (2 nodos pSeries 510):
2 procesadores 2.5GHz dual-core PowerPC 970MP	2 procesadores 2.5GHz dual-core PowerPC 970MP	1 procesador 2.1GHz dual-core Power5+
8GB 533MHz DDR2 SDRAM	8GB 533MHz DDR2 SDRAM	4GB 533MHz DDR2 SDRAM
Disco SAS 73GB	Disco 73GB SAS	Disco 373GB SAS
1 x Myricom M3S-PCIXD-2-I	1 x Myricom M3S-PCIXD-2-I	2 x Emulex LP10000 PCI-X/133MHz FC adapters

Tabla 1 – Información de los nodos del supercomputador atlante

PC

En lo concerniente al análisis de las especificaciones técnicas del equipo de trabajo en el que se realizará la programación del algoritmo paralelo, cabe destacar que no se ha realizado un estudio en profundidad sino más bien, sólo acerca de las características pertinentes al proyecto.

Dicho equipo presenta un procesador Intel Core i7-2670QM (quad-core, o, de cuatro núcleos) cuya frecuencia oscila entre los 2200 y los 3100 MHZ. Además, ofrece Hyperthreading para controlar 8 hilos de manera simultánea para un uso mejorado del pipeline. [11]

Rendimiento de la GPU

La NVIDIA GeForce GT 540M es una tarjeta gráfica rápida de clase media para ordenadores portátiles. Se basa en el mismo chip que la GeForce GT 435M, pero con velocidades de reloj ligeramente más altos. Además, es compatible con GDDR5 y DDR3 900MHz.

Tiene 96 núcleos de procesamiento y una interfaz de memoria de 128-bit soporte GDDR5 y DDR3. Mientras el reloj gráfica es máximo a 672MHz, el procesador lo es a 1344MHZ y la memoria tiene una velocidad de hasta 900MHz. [12]

Soporta CUDA, OpenCL y DirectCompute 2.1.

Resumen

Especificaciones de motor de GPU:	
CUDA Cores	96
Reloj de procesador (MHz)	1344 MHz
Tasa de relleno de texturas (miles de millones/s)	10.8

Tabla 2 – Especificaciones de motor de GPU

Especificaciones de memoria:	
Reloj de la memoria (MHz)	900
Config. de memoria estándar	DDR3
Interfaz de memoria	128-bit
Ancho de banda de memoria (GB/s.)	28.8

Tabla 3 – Especificaciones de memoria

Situación del software

De igual manera que en el apartado 4.1, antes de pasar a detallar de manera exhaustiva el estado actual del software, resulta útil conocer ciertos conceptos.

Para poder evaluar el desempeño, hay que definir y medir su rendimiento por lo que se debe determinar qué factores influyen en la ejecución y así precisar una expresión que lo caracterice.

Resulta evidente que la medida principal del rendimiento de un algoritmo es el tiempo. Aquel que se realice su cometido en el menor tiempo es el que presenta mejores prestaciones (suponiendo que ambos funcionan sobre la misma máquina).

Un algoritmo secuencial es evaluado por su tiempo de ejecución como función del tamaño del problema.

En cambio, en el caso de un programa paralelo y/o concurrente depende del tamaño del problema, del número de procesadores y de ciertos parámetros de comunicación de la plataforma. Es por ello que dichos algoritmos deben ser valorados y analizados teniendo en cuenta también la plataforma.

Por otro lado, la escalabilidad es un factor significativo dado que se dice que un sistema es escalable si mantiene constante la eficiencia al aumentar el número de procesadores aumentando también el tamaño de la muestra.

En este campo, el del flujo óptico, también hay que añadir el concepto "endpoint error" o error al calcular la distancia entre píxeles entre dos frames.

Sabiendo esto, a continuación, se presentan sólo los primeros resultados (tanto a lo ancho como a lo largo) de las tablas (solamente con los resultados promedio dado que son bastante extensas) donde se exponen en orden creciente, según la magnitud del error cometido, una evaluación de la calidad de la precisión e interpolación de los distintos algoritmos de detección de flujo óptico.

Estas tablas muestran enlaces a los resultados actuales donde se exhiben dos medidas de precisión del flujo (punto final y error angular) y dos medidas de calidad de interpolación. Para cada una de las 4 medidas, se reportan 8 indicadores de error resultando 32 tablas.

Debe hacerse hincapié en que los errores de punto final han de considerarse como la medida más adecuada de precisión del algoritmo. No obstante, no existe ninguna métrica de error estándar.

Tabla 4 – Error promedio de punto final

Average endpoint error	avg. Rank	Army			Mequon		
		(Hidden texture)			(Hidden texture)		
		GT	im0	im1	GT	im0	im1
		all	Disc	untext	all	disc	untext
ComplexFlow [81]	2.6	0.07 1	0.20 2	0.05 1	0.15 1	0.51 3	0.12 5
OFLADF [82]	6.7	0.08 7	0.21 3	0.06 5	0.16 5	0.53 4	0.12 5
MDP-Flow2 [70]	7.3	0.08 7	0.21 3	0.07 14	0.15 1	0.48 1	0.11 1
NN-field [73]	8.2	0.08 7	0.22 13	0.05 1	0.17 7	0.55 6	0.13 10
Epistemic [84]	9.2	0.07 1	0.21 3	0.05 1	0.16 5	0.55 6	0.12 5
TC/T-Flow [80]	14.0	0.07 1	0.21 3	0.05 1	0.19 13	0.68 24	0.12 5
LME [72]	15.2	0.08 7	0.22 13	0.06 5	0.15 1	0.49 2	0.11 1
ADF [67]	15.3	0.08 7	0.22 13	0.06 5	0.18 9	0.62 15	0.14 15
Layers++ [37]	15.5	0.08 7	0.21 3	0.07 14	0.19 13	0.56 8	0.17 25
IROF++ [58]	16.1	0.08 7	0.23 17	0.07 14	0.21 25	0.68 24	0.17 25
nLayers [57]	16.5	0.07 1	0.19 1	0.06 5	0.22 30	0.59 10	0.19 40

[13]

Tabla 5 - Error promedio de ángulo

Average angle error	avg. Rank	Army (Hidden texture)			Mequon (Hidden texture)		
		GT	im0	im1	GT	im0	im1
		all	Disc	untext	All	disc	untext
	rank	all	Disc	untext	All	disc	untext
ComplexFlow [81]	3.4	2.69 1	7.56 2	1.98 2	1.97 3	7.01 3	1.59 4
NN-field [73]	7.0	2.89 6	8.13 12	2.11 4	2.10 5	7.15 6	1.77 12
OFLADF [82]	8.5	3.04 13	7.80 6	2.40 12	2.14 6	7.02 4	1.72 8
nLayers [57]	11.3	2.80 4	7.42 1	2.20 6	2.71 21	7.24 7	2.55 41
MDP-Flow2 [70]	12.2	3.23 24	7.93 9	2.60 17	1.92 1	6.64 1	1.52 1
Epistemic [84]	12.4	2.78 3	8.20 13	2.05 3	2.04 4	7.31 8	1.66 7
TC/T-Flow [80]	15.0	2.69 1	7.75 5	1.87 1	2.76 24	10.2 30	1.73 9
ADF [67]	15.8	2.98 10	8.32 16	2.28 7	2.27 9	8.35 16	1.81 13
FC-2Layers-FF [77]	16.9	3.02 12	7.87 8	2.61 18	2.72 22	9.35 24	2.29 28
Layers++ [37]	18.0	3.11 14	8.22 14	2.79 29	2.43 14	7.02 4	2.24 23

[14]

Tabla 6 - Error promedio de interpolación

Average angle error	avg. Rank	Army (Hidden texture)			Mequon (Hidden texture)		
		GT	im0	im1	GT	im0	im1
		all	disc	untext	All	disc	untext
	rank	all	Disc	untext	All	disc	untext
MDP-Flow2 [70]	10.0	2.89 5	5.38 5	1.19 1	3.47 7	5.07 7	1.26 1
CBF [12]	17.0	2.83 1	5.20 1	1.23 39	3.97 37	5.79 32	1.56 38
Deep-Matching [85]	18.3	3.02 18	5.68 17	1.23 39	3.92 36	5.80 33	1.56 38
SuperFlow [89]	18.8	2.94 8	5.56 12	1.24 45	3.99 38	5.78 30	1.67 54
Aniso. Huber-L1 [22]	20.0	2.95 10	5.44 7	1.24 45	4.42 58	6.27 57	1.67 54
CLG-TV [48]	20.5	2.94 8	5.45 8	1.25 53	4.26 49	6.17 48	1.60 44
NN-field [73]	21.3	2.98 13	5.70 18	1.20 5	3.31 3	4.73 3	1.26 1
ComplexFlow [81]	21.3	2.92 7	5.51 11	1.19 1	3.30 2	4.71 2	1.26 1
IROF-TV [53]	21.8	3.07 27	5.91 33	1.23 39	3.71 21	5.47 18	1.40 22
LME [72]	22.0	2.95 10	5.59 14	1.19 1	3.68 18	5.50 21	1.38 19

[15]

Tabla 7 - Interpolación normalizada

Average angle error	avg. Rank	Army (Hidden texture)			Mequon (Hidden texture)		
		GT	im0	im1	GT	im0	im1
		All	Disc	Untext	All	disc	untext
	rank	All	Disc	Untext	All	disc	untext
NN-field [73]	15.2	0.59 2	0.77 11	0.64 3	0.59 1	0.77 3	0.58 1
MDP-Flow2 [70]	15.6	0.59 2	0.72 2	0.63 1	0.62 7	0.85 8	0.58 1
SuperFlow [89]	19.8	0.62 31	0.84 43	0.66 45	0.76 37	1.04 37	0.69 54
ALD-Flow [68]	20.0	0.62 31	0.81 30	0.66 45	0.70 27	0.99 28	0.62 25
ComplexFlow [81]	20.8	0.58 1	0.71 1	0.63 1	0.59 1	0.76 1	0.58 1
Deep-Matching [85]	22.5	0.63 45	0.85 47	0.65 25	0.75 36	1.04 37	0.67 47
LME [72]	22.8	0.59 2	0.72 2	0.64 3	0.66 16	0.90 16	0.62 25
ADF [67]	23.7	0.59 2	0.73 4	0.64 3	0.68 24	0.97 24	0.62 25
CLG-TV [48]	24.2	0.63 45	0.86 49	0.66 45	0.81 50	1.12 50	0.66 42
IROF++ [58]	24.7	0.59 2	0.74 5	0.64 3	0.65 14	0.89 12	0.59 4

[16]

Las tablas presentadas previamente proporcionan una instantánea del estado del arte del flujo óptico. Todos son de diferentes autores.

En todas se compara el rendimiento de los algoritmos versus los errores producidos en cada una de las cuatro medidas: error final (Endpoint error), error angular (EA), error de interpolación (EI) y el error de interpolación normalizado (NE) en una secuencia de pruebas de 8 tests.

Además se listan los tiempos de ejecución, en segundos, que han sido brindados por los autores, (no se han añadido a este documento) aunque no han sido normalizados (entorno de programación, velocidad de CPU, cantidad de núcleos, etc.)

La columna (a) describe el rango medio calculado en siete de las ocho estadísticas (la desviación estándar está omitida) y las tres máscaras del error del punto final.

La columna (b) contiene el correspondiente rango promedio para el error angular.

La columna (c) muestra el rango promedio para cada una de las siete estadísticas del error del punto final calculado con los ocho conjuntos de datos.

La columna (d) presenta el error medio de punto final (endpoint) para cada una de las tres máscaras calculado sólo con los ocho conjuntos de datos.

La columna (e) tiene el promedio del error de punto final (Endpoint error) de cada conjunto de datos.

Los algoritmos de la parte superior de la tabla tienden a usar más mejoras sobre datos y términos anteriores además de ponderación espacial e isotrópica.

Optimización continua: los algoritmos que utilizan el método del descenso del gradiente, aparecen en la parte inferior de la tabla. Por otra parte, los métodos variacionales aparecen a lo largo de esta. Hay que tener en cuenta que existe una correlación entre el uso de métodos variacionales y las funciones de energía más sofisticadas que no es intrínseca al enfoque variacional.

Optimización discreta: Los algoritmos de optimización discreta no funcionan muy bien. Nótese, sin embargo, que las funciones de energía usadas en estos métodos son en general relativamente simples y pueden ser extendidos en el futuro para incorporar algunos de los elementos más sofisticados.

Flujo óptico

El cálculo del flujo óptico consiste en la estimación del movimiento aparente de los objetos en una secuencia. Dado un conjunto de imágenes, el objetivo es calcular el desplazamiento de los píxeles entre cada una.

Podremos encontrar una serie de objetos estáticos o dinámicos que, por lo general, se verán influenciados por condiciones variables del entorno, tales como fuentes de iluminación, sombras, reflejos y otros efectos luminosos, así como por otras dificultades asociadas a la aparición y desaparición de objetos en la escena o la oclusión de unos con otros.

Cuando se comenzó a profundizar en este campo, todas las investigaciones realizadas estaban estrechamente relacionadas con el ámbito de la robótica. No obstante, cuando los ordenadores personales fueron accesibles a la gente de a pie, a la par que incrementaron estos su capacidad de cómputo, se creó una situación propicia a la que la visión artificial podía dar respuesta a esta clase de problemas. Durante todo este tiempo se han desarrollado métodos que ofrecen soluciones de gran precisión. Sin embargo, todavía hay lugar para la mejora, como, por ejemplo, la que se intenta abordar con este proyecto.

La importancia de la estimación del flujo radica fundamentalmente en el gran número de aplicaciones que tiene, como por ejemplo, reconstrucción 3D, compresión de video, segmentación, detección de objetos, sistemas de navegación robótica, etc.

En nuestro caso queremos conocer el movimiento que se registra en la escena lo que provoca que sea un problema mal condicionado [17] ya que puede haber varias soluciones para un mismo desplazamiento, lo que da lugar a cierta ambigüedad. Para revertir este inconveniente y convertirlo en bien condicionado se tiene que cumplir tres condiciones: (i) existencia, (ii) unicidad y (iii) estabilidad de la solución. Para cumplir con estos requisitos se suele recurrir a la incorporación de un elemento de estabilización, normalmente en forma de término de regularización o suavizado.

Para detectar las correspondencias de los píxeles entre dos imágenes se suele suponer que alguna propiedad de la imagen no varía a lo largo del tiempo. Esta suposición se puede representar tal que:

$$f(x+u, y+v) - f(x,y) = 0$$

donde f es algún tipo de propiedad en la imagen.

La intensidad de los píxeles de una imagen es un valor que indica la cantidad de radiación luminosa reflejada por la superficie de los objetos. Existen muchos modelos para representar los distintos tipos de superficies. Uno de los más simples es el modelo que considera que el brillo aparente es el mismo para todas las direcciones de vista. Por tanto, si se asume esto, basta con sustituir f por I.

Partiendo de la expresión no lineal $f(x+u, y+v) - f(x,y) = 0$ y aplicando un desarrollo de Taylor se obtiene la ecuación que ha sido bautizada como la ecuación de restricción del flujo óptico.

$$I_x U + I_y V = 0$$

donde los subíndices indican derivadas parciales.

Clasificación de los métodos

A pesar de las muchas contribuciones que ha habido en este campo, todo el trabajo realizado se podrá resumir en unos pocos métodos de referencia. El resto de contribuciones supone la aportación de nuevos elementos que, combinados con los anteriores, han permitido mejorar la precisión de las estimaciones. Dado el gran volumen de contribuciones en este ámbito, algunos autores han propuesto una serie de clasificaciones para establecer relaciones entre los distintos métodos.

Métodos diferenciales

Calculan el desplazamiento de los píxeles a partir de las derivadas espaciales de las intensidades de la imagen.

Métodos globales

Usan la ecuación $I_x U + I_y V = 0$ y añaden como restricción global un término de regularización sobre el flujo que supone que el campo de desplazamiento es suave. Obtienen campos de desplazamiento densos.

Métodos locales

Emplean la información de los píxeles circundantes para estimar su movimiento. El método más representativo y destacado es el de Lucas-Kanade. El cual calcula el desplazamiento a partir de la minimización de la ecuación del flujo óptico alrededor de una ventana centrada en un píxel.

El mayor inconveniente es que sólo es posible detectar el movimiento en aquellas zonas donde existan variaciones en la imagen. En zonas homogéneas donde puede haber movimiento este no es detectable. Por ello, los campos de desplazamiento no son densos.

Métodos de superficie

Los métodos pertenecientes a esta gama realizan una segmentación del flujo óptico de los objetos (o superficies) que se mueven independientemente. Los métodos de contornos utilizan la información de los bordes de los objetos para detectar el desplazamiento. Aplican técnicas diferenciales para la extracción de determinadas estructuras en la imagen para luego establecer correspondencias entre estas estructuras.

Métodos basados en la correlación

Realizan la búsqueda de correspondencias mediante el empleo de ventanas o patrones alrededor de cada pixel. La idea sobre la que se basan estos métodos radica en la sencillez de encontrar correspondencias entre los píxeles a través de la comparación de regiones por medio de la maximización de alguna característica común. Una ventaja que tienen estos métodos es que al usar mayor información la búsqueda de las correspondencias es más efectiva.

Métodos basados en la frecuencia

Usan las transformadas de Fourier para calcular el flujo óptico a través del dominio de la frecuencia.

Métodos variacionales

La base de este tipo de procedimientos es la definición de una energía que penaliza las desviaciones respecto a las restricciones impuestas en el modelo. Una de las ventajas que ofrecen es que todas estas restricciones están presentes en la energía; no existe ningún tipo de suposición adicional que no se refleje en el modelo y si en la implementación. A diferencia de otro tipo de

técnicas las estimaciones son densas por lo que no es necesario realizar ningún proceso de interpolación.

En este proyecto se aborda el caso particular de un método variacional simple en el que se utiliza únicamente un sólo canal, específicamente un imagen en escala de grises:

$$D(\nabla I_1) = \frac{1}{\|\nabla x\|^2 + 2\zeta^2}(\xi\xi^t + \zeta^2 Id)$$

O,

$$D(\nabla I_1) = \frac{1}{\|\nabla x\|^2 + 2\zeta^2}\left(\begin{bmatrix} \frac{dI_1^2}{dy} & \frac{-dI_1}{dy}\frac{dI_1}{dx} \\ \frac{-dI_1}{dy}\frac{dI_1}{dx} & \frac{dI_1^2}{dx} \end{bmatrix} + \zeta^2 Id\right)$$

$\xi = \left(\frac{\partial I_1}{\partial y}, \frac{-\partial I_1}{\partial x}\right)^t$ es un vector ortogonal a ∇I_1, Id es la matriz identidad y lambda es un coeficiente que determina el comportamiento isotrópico del suavizado e inhibe los bordes borrosos cuando la magnitud del gradiente es alta, es decir mucho mayor que lambda.

Los parámetros de entrada utilizados son C y $\lambda \in (0, 1)$. A partir de estas variables se calculan dos pesos utilizados en el modelo de energía como son C y ζ donde :

$$\alpha = \frac{\mathrm{C}}{max(\|(\nabla G_\sigma * I_1)(\bar{x})\|^2)}$$

$$\lambda = \int_0^{\zeta} H_{\|(\nabla G_\sigma * I_1)(\bar{z})\|} dz$$

Donde $G_\sigma * I_1$ representa la convolución de I_1 con una gaussiana de desviación estándar σ, $H_{\|(\nabla G_\sigma * I_1)(\bar{z})\|}dz$ representa el histograma normalizado de $(\|\nabla G_\sigma * I_1\|)$. λ Se conoce como la fracción isotrópica. Cuando tiende a cero, la

difusión aplicada es anisotrópica mientras que cuando tiende a uno se hace isotrópica. Esta normalización depende de α y ζ permite a la energía ser invariante frente a cambios de intensidad.

Secuencias de prueba

En esta sección se hace una breve descripción de las distintas secuencias tanto reales como sintéticas que se han utilizado para la evaluación del método de flujo óptico.

Por un lado, tienen por objetivo cubrir todos los tipos de situaciones que se pueden dar en cualquier secuencia y, por otro lado, la estandarización de las secuencias facilita las comparaciones entre los distintos métodos.

Las secuencias sintéticas permiten conocer con total exactitud el desplazamiento de los objetos presentes en la escena. De este modo, podemos evaluar la diferencia entre el movimiento exacto y el estimado. Sin embargo, en este tipo de secuencias no suelen aparecer determinadas perturbaciones que si hay en el mundo real. Por este motivo, también se han utilizado secuencias reales para analizar el comportamiento de los métodos ante distinto tipo de perturbaciones.

Secuencias sintéticas

Secuencia del cuadrado

En este par de frames puede apreciarse en la primera imagen cuatro cuadrados casi perfectamente alineados en sus correspondientes esquinas mientras que cuando miramos al segundo podremos ver que han sufrido los siguientes desplazamientos:

- esquina superior izquierda (10, 5)
- esquina superior derecha (-10, 0)
- esquina inferior izquierda (0, -5)
- esquina inferior derecha (-10, -10)

Marbel Blocks

Esta secuencia está compuesta por treinta imágenes de tamaño 512 x 512 píxeles en la que aparecen cinco torres de mármoles, el 80% de las cuales (las oscuras) permanecen estáticas en la escena mientras que la restante se desplaza hacia la izquierda.

La cámara que capta la escena se traslada hacia la izquierda por lo que todos los objetos estáticos aparecen con un movimiento hacia la derecha, excepto, el bloque de mármol blanco que también se mueve hacia la izquierda pero a una velocidad mayor a lo que lo hace la cámara.

Yosemite

Está compuesta por quince imágenes de tamaño 316 x 252 píxeles. En esta secuencia se combinan movimientos divergentes.

En esta zona de la imagen el desplazamiento es continuo y constante. En el resto de la imagen, el movimiento es muy suave creándose un efecto de zoom que simula un acercamiento al valle.

Secuencias reales

A diferencia de las secuencias sintéticas, estas presentan una serie de perturbaciones, como efecto de entrelazado, cambios de iluminación, sombras, etc., que dificultan la correcta estimación del flujo.

Secuencia del taxi de Hamburgo

La secuencia del Taxi de Hamburgo se trata de una de las más famosas y más utilizadas en los trabajos sobre el cálculo del flujo óptico.

En la esquina inferior derecha, circula un camión justo detrás del taxi y, por último, en la esquina superior izquierda un peatón se desplaza por la acera a una velocidad muy inferior a la que lo hacen los vehículos. El resto de objetos presentes en la secuencia no se mueven.

Estudio de herramientas

Para valorar de forma realista las necesidades que se pueden cubrir con este proyecto, así como aquellos aspectos en los que se puedan innovar, se ha realizado un estudio de algunas de las herramientas disponibles que permitan el mejor desarrollo posible así como la mejor plataforma de investigación.

Este estudio se halla descrito en el apartado 4.1 dado que las herramientas necesarias para realizar esta investigación son el hardware en si descrito en dicho apartado.

Conclusiones

En cuanto al hardware podemos afirmar taxativamente que existen opciones útiles que elegiremos para este proyecto por cuanto cuentan con las características necesarias para cubrir los objetivos.

No obstante, en lo que al software se refiere, a pesar de la gran variedad de algoritmos de flujo óptico presentados con anterioridad, no se ha encontrado una versión paralela que estime densos campos de flujo con gran desplazamiento, de manera fiable. Es por ello que se propone el desarrollo de una herramienta genérica multiplataforma que permita ejecutar dicho algoritmo de forma paralela y que aproveche toda la capacidad de cómputo ofrecida por las arquitecturas multi-core y por las GPU modernas.

Se pretende analizar, diseñar e implementar una interfaz única de escritorio, multiplataforma, basada en Qt, que permita modificar ciertos parámetros de entrada.

Metodología

Seguidamente se explican las herramientas de análisis que han sido utilizadas para el desarrollo del proyecto y su implementación.

Modelo de proceso de software

La metodología empleada en el desarrollo software del proyecto ha sido el modelo en espiral, propuesto originalmente por Boehm en 1988, el cual es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. [18]

Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.

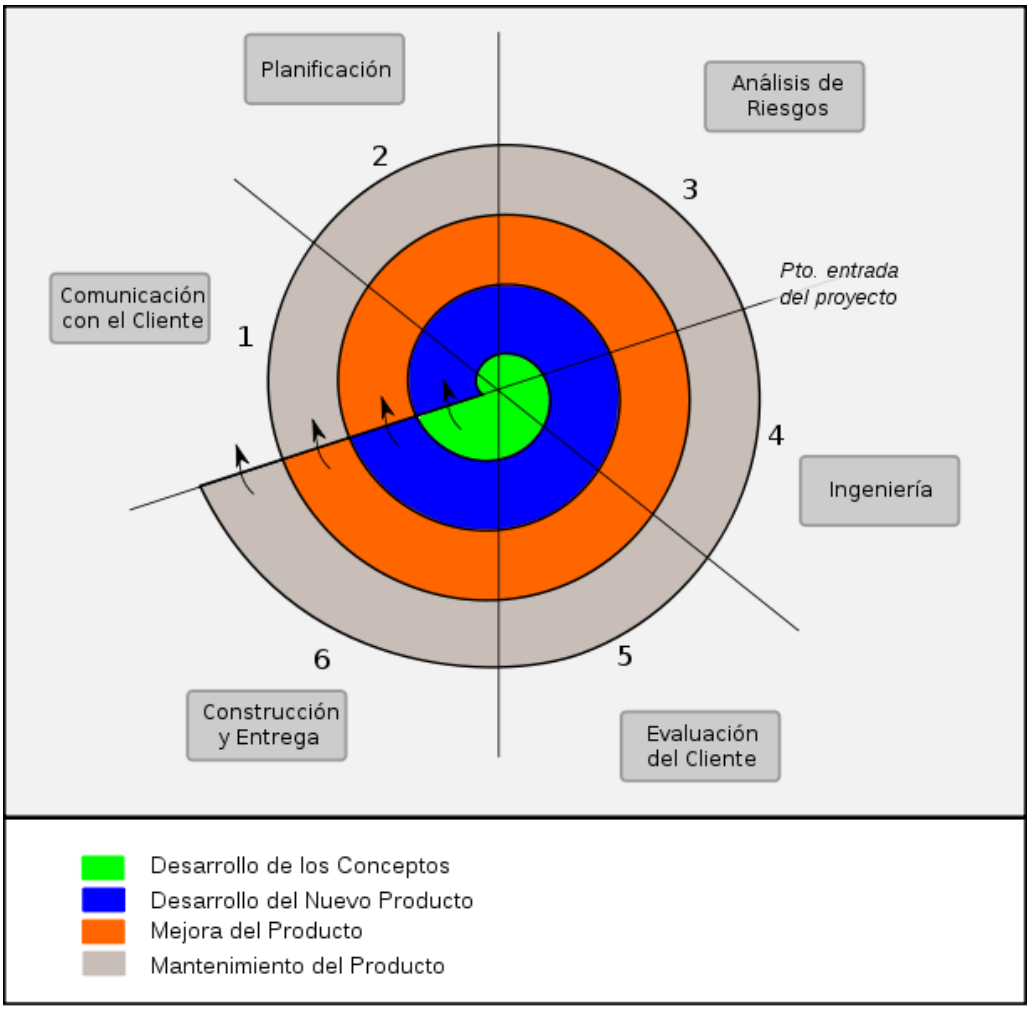


Imagen 2 - Fases del modelo en espiral

El modelo en espiral se divide en un número de actividades de marco de trabajo, también llamadas regiones de tareas. Generalmente, existen entre tres y seis regiones de tareas. El modelo en espiral contiene seis regiones de tareas:

- **Comunicación con el cliente:** se trata de las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.
- **Planificación**: los pasos necesarios para definir recursos, el tiempo y otra información relacionadas con el proyecto.
- **Análisis de riesgos**: las requisitos exigidos para evaluar riesgos técnicos y de gestión.
- **Ingeniería**: las procedimientos demandados para construir una o más representaciones de la aplicación.
- **Construcción y acción**: las elementos obligatorios para construir, probar, instalar y proporcionar soporte al usuario, como documentación y práctica.

- **Evaluación del cliente**: las labores imprescindibles para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.

Cuando empieza este proceso evolutivo, se gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. El primer circuito de la espiral puede producir el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para producir un prototipo y progresivamente versiones más sofisticadas del software. Cada paso por la región de planificación produce ajustes en el plan del proyecto. El coste y la planificación se ajustan con la realimentación ante la evaluación del cliente.

El modelo en espiral es un enfoque realista de la elaboración de sistemas y de software a gran escala. Como el software evoluciona, a medida que progresa el proceso, el ingeniero y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos. El modelo en espiral utiliza la construcción de prototipos como mecanismo de reducción de riesgos, pero, lo que es más importante, permite aplicar, a quien lo prepara, el enfoque de construcción de prototipos en cualquier etapa de la evolución. Mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo iterativo que refleja de forma más realista el mundo real.

Lenguaje de modelado

El lenguaje de modelado utilizado a lo largo del proyecto será UML o Lenguaje Unificado de Modelado. Permite expresar un patrón de análisis utilizando una notación de prototipado con unas reglas sintácticas, semánticas y prácticas.

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un ejemplo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, además de aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Se emplea para definir un sistema, para detallar los artefactos en el mismo, para documentar y construir.

UML no puede compararse con la programación estructurada, pues es solamente un lenguaje de estereotipado; sólo se diagrama la realidad de una utilización en un requisito.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

Metodología aplicada

El proyecto descrito en este documento se desarrollará en base al modelo de proceso de software y el lenguaje de modelado indicados. A la hora de analizar y diseñar el sistema a desarrollar, se emplearán los diagramas elaborados siguiendo las pautas establecidas por el Lenguaje Unificado de Modelado UML. Entre estos diagramas destacan la identificación de actores, casos de uso, de clases y de secuencia.

La producción del software se llevará a cabo utilizando el Modelo de Proceso en Espiral. Se irán generando prototipos de validación tras pasar por cada una de las fases descritas en el modelo, sobre todo en las tareas fundamentales de análisis, diseño e implementación.

Recursos necesarios

En esta sección se enumeran los recursos hardware y software que han sido necesarios para abordar este proyecto. Cabe destacar que todos son orientados al software libre.

Recursos Hardware

PC

Para poder ejecutar la aplicación y poder comprobar los resultados provenientes de la paralelización del algoritmo será necesario contar con un sistema informático que disponga de una tarjeta gráfica NVIDIA que soporte CUDA.

Recursos Software

Sistema Operativo

Debido a la naturaleza del proyecto – paralelización de un algoritmo – se puede optar por cualquier sistema operativo. Sin embargo, se da el caso de que en equipo de desarrollo se usará Ubuntu 14.10. A pesar de todo, no se incurre en ningún problema porque CUDA Y Qt son multiplataforma.

Editor de código y compiladores - IDE

El sistema que se va a desarrollar se trata de una aplicación de escritorio. Es por ello que será necesario poseer un software para la edición, depuración y compilación avanzada de proyectos en el lenguaje C++, con el fin de agilizar la tarea de programación de la aplicación y su interfaz.

Para llevar a cabo estas tareas se ha instalado en la máquina de desarrollo el entorno de desarrollo integrado Eclipse en su versión “Helios”

junto con el depurador Gdb (GNU gdb (GDB) 7.3.50.20111026-cvs (cygwin-special)) y los compiladores Gcc (4.5.2) y G++ (4.5.2) y la biblioteca multiplataforma Qt (4.8) para desarrollar la interfaz gráfica.

El IDE Eclipse es un entorno de desarrollo integrado, libre y gratuito sin restricciones de uso; una herramienta para pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos que permiten su extensión, entre los cuales se encuentra "Toolchain for the NVIDIA CUDA nvcc compiler" que permitirá el desarrollo de la programación para CUDA.

Además, gracias a sus herramientas de autocompletado de código, agiliza el proceso de programación y ayuda a la correcta estructuración del mismo.

Editor de UML

La gran mayoría de los diagramas elaborados a lo largo de las diferentes fases del proyecto serán elaborados mediante UML, por lo que será necesario emplear alguna herramienta de edición que permita trabajar de forma sencilla y rápida con este lenguaje.

En concreto, se ha utilizado el software denominado StarUML. Se trata de una herramienta de software libre de edición UML, bajo licencia modificada de GNU GPL. Es compatible con la mayoría de los tipos especificados en el diagrama de UML 2.0 y fue desarrollada en Delphi.

Editor de textos

A lo largo de todo el proceso ha sido necesario documentar las diferentes fases del proyecto, así como generar manuales de ayuda y de uso de cara al usuario final. Para ello es necesario utilizar herramientas de edición de textos, en concreto se ha hecho uso de Microsoft Office Word para llevar a cabo toda la documentación y guías necesarias.

Git

Git es un sistema de control de versiones distribuido, libre y de código abierto diseñado para manejar desde pequeños repositorios hasta grandes proyectos con rapidez y eficiencia.

Destaca principalmente por su modelo de ramificación. En Git las ramas, o líneas de desarrollo pueden ser totalmente independientes entre sí aunque pueden fusionarse. Esto permite realizar experimentos o centrarse en nuevas características de la aplicación.

Valgrind

Es un marco de instrumentación para la construcción de herramientas de análisis dinámicos. Presenta un conjunto de herramientas que pueden detectar automáticamente muchos errores de gestión de la memoria, y realizar un perfil de sus programas en detalle.

Cuda-memcheck

Cuda-memcheck permite identificar con precisión el origen y la causa de los errores de acceso a memoria en un entorno multi-hilo. Además detecta estos errores en el código de GPU y permite localizarlos rápidamente.

Herramientas externas

Bitbucket

Bitbucket es un servicio de alojamiento de código basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Ofrece planes repositorios privados incluso en sus planes gratuitos y permite trabajar en equipo.

Tecnologías utilizadas

Para llevar a cabo el desarrollo e implementación de la paralelización del algoritmo se han utilizado una serie de tecnologías específicas.

C++

C++ es un lenguaje de programación compilado y orientado a objetos. Es imperativo y de tipado estático.

Sus usos son múltiples y de propósito general. Sus campos de aplicación incluyen software de sistemas, de aplicaciones, controladores de dispositivos, software integrado, servidor de alto rendimiento y software de entretenimiento como los videojuegos.

Qt

Qt es una biblioteca multiplataforma, de software libre y código abierto, ampliamente utilizada para desarrollar aplicaciones con interfaz gráfica. Utiliza el lenguaje de programación C++ de forma nativa.

Ha sido desarrollada a través de Qt Project, donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas.

Cuda

CUDA, Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo), es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico), utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un altísimo número de hilos simultáneos, para proporcionar un incremento extraordinario del rendimiento del sistema.

Análisis

Tras realizar el estudio de la situación actual y habiendo visto que no existen aplicaciones para entornos gráficos que calculen el flujo óptico y muestren los resultados, se pasa a detallar la línea de trabajo necesaria para desarrollar el proyecto:

- Interfaz gráfica
- Formatos de entrada y salida eficientes
- Información general
- Área de resultados

No obstante, la fase de análisis va a permitir identificar y definir formalmente los requerimientos de la aplicación antes de pasar a su diseño e implementación. Dichos requisitos son, según Pressman, la descripción de los servicios proporcionados por el sistema que se pretende desarrollar y sus restricciones operativas

Introducción

Se ha puesto especial interés en esta sección, pues una documentación bien estructurada y detallada facilita tanto el trabajo que se pretende desarrollar actualmente como el futuro por cuanto es esencial comprender perfectamente los requisitos del software.

Objetivo

El fin de esta sección es ayudar en el proceso de decisión a la hora de dejar claro qué es lo que se va a construir. Se busca enumerar los requisitos técnicos incluyendo todas las interfaces con humanos y otros sistemas.

Estructura

Hay que tener en cuenta que la aplicación, aunque esté orientada a un público técnico, debe poseer una presentación totalmente intuitiva y fácil de usar.

Por lo tanto, parece apropiado crear una estructura en la que los usuarios sepan en todo momento si están procesando dos frames individuales o un conjunto perteneciente a un video a la par que dispongan de métodos informativos que les permitan conocer información acerca de las imágenes, de la tarjeta gráfica que se está usando en el momento del cálculo y los resultados finales, una vez haya acabado el cómputo del flujo óptico.

Contenido

Teniendo en cuenta el apartado anterior, los contenidos de la vista principal de la ventana serán dos etiquetas que muestren el nombre junto con la ruta de las imágenes cargadas para su proceso en el programa. Una tercera etiqueta que muestre la ruta de la secuencia de vídeo si así lo decide el usuario.

A continuación, debajo, un contenedor para cada par de frames procesados con su correspondiente frame resultante que será la consecuencia de sumarle el resultado del flujo óptico a la primera imagen. Cuánto más parecido a la segunda imagen, mejor habrá sido el cálculo.

Siguiendo con esta filosofía, en el menú principal tendremos 4 opciones en las que se brindarán ventanas emergentes que mostrarán información sobre la tarjeta gráfica, los resultados finales, las características de los frames, etc.

Requisitos del software

Identificación de actores

Para poder elaborar el modelo de casos de uso, se determinarán diferentes aspectos relevantes. Por un lado, se deberán identificar los actores

del sistema así como su papel en el mismo. Por otro lado, se indicarán cada una de sus acciones correspondientes.

El siguiente diagrama muestra los actores del sistema.

Imagen 3 - Identificación de actores

Actor	Tipo	Definición
Usuario	Principal	Es el único actor del sistema, al que se le permite realizar todas las operaciones disponibles.

Tabla 8 - Definición de actores

Acciones

Actor	Acción	Resumen
Usuario	Cargar frames	Podrá seleccionar dos imágenes aleatorias para procesar.
	Cargar Video	Podrá seleccionar una secuencia de vídeo para tratar a lo largo de cada par de frames
	Procesar frames con CPU	Podrá ejecutar el algoritmo en la CPU
	Procesar frames con GPU	Podrá poner en marcha el algoritmo en la GPU
	Examinar frames	Podrá conocer las propiedades de las imágenes seleccionadas
	Examinar la tarjeta gráfica	Podrá examiar las características de la/s tarjeta/s gráfica/s presentes en la máquina.
	Ver los resultados	Tras la finalización del algoritmo será capaz de examinar y navegar por los resultados numéricos y no numéricos.

Tabla 9 - Acciones de los actores

Es excesivamente simple dado que no existe ninguna jerarquía como podría ser, por ejemplo, administrador, súper usuario, Jefe de departamento, etc. Ni tampoco se requiere un proceso de autenticación por lo que tampoco existe diferencia entre usuarios registrados / autenticados o no.

Diccionario de conceptos

Además de identificar los actores del sistema, así como cada una de las acciones que pueden realizar en el mismo, se presentan a continuación una serie de conceptos que deben ser manejados por la aplicación:

- Frame: cada uno de los fotogramas individuales o dentro de una sucesión de imágenes que componen un vídeo.

- Hilo (Thread): es la ejecución de un núcleo (kernel) con un índice dado. Cada hilo utiliza su índice para acceder a los elementos de matriz de tal manera que la colección de todos los hilos cooperan para procesar todo el conjunto de datos

- Bloque: Conjunto de hilos.

- Grid: Conjunto de bloques

- Kernel (núcleo): es una función la cual al ejecutarse lo hará en N distintos hilos en lugar de en secuencial.

- Matriz escasa: es una matriz caracterizada porque la mayoría de sus elementos son cero.

- Ruleta de color: en una imagen de referencia por medio de la cual podemos crear, en la aplicación, una imagen en la que cada pixel adquiere un color característico según la orientación y la amplitud del flujo tomando como punto de partida el centro de dicha imagen.

Modelo de casos de uso

Una vez identificados los actores del sistema, se podrá elaborar el diagrama general de casos de uso, así como la explicación detallada de cada uno.

En el siguiente diagrama se presentan cada uno de los casos de uso del sistema a desarrollar, que serán desglosados y explicados detalladamente a continuación en forma de tablas.

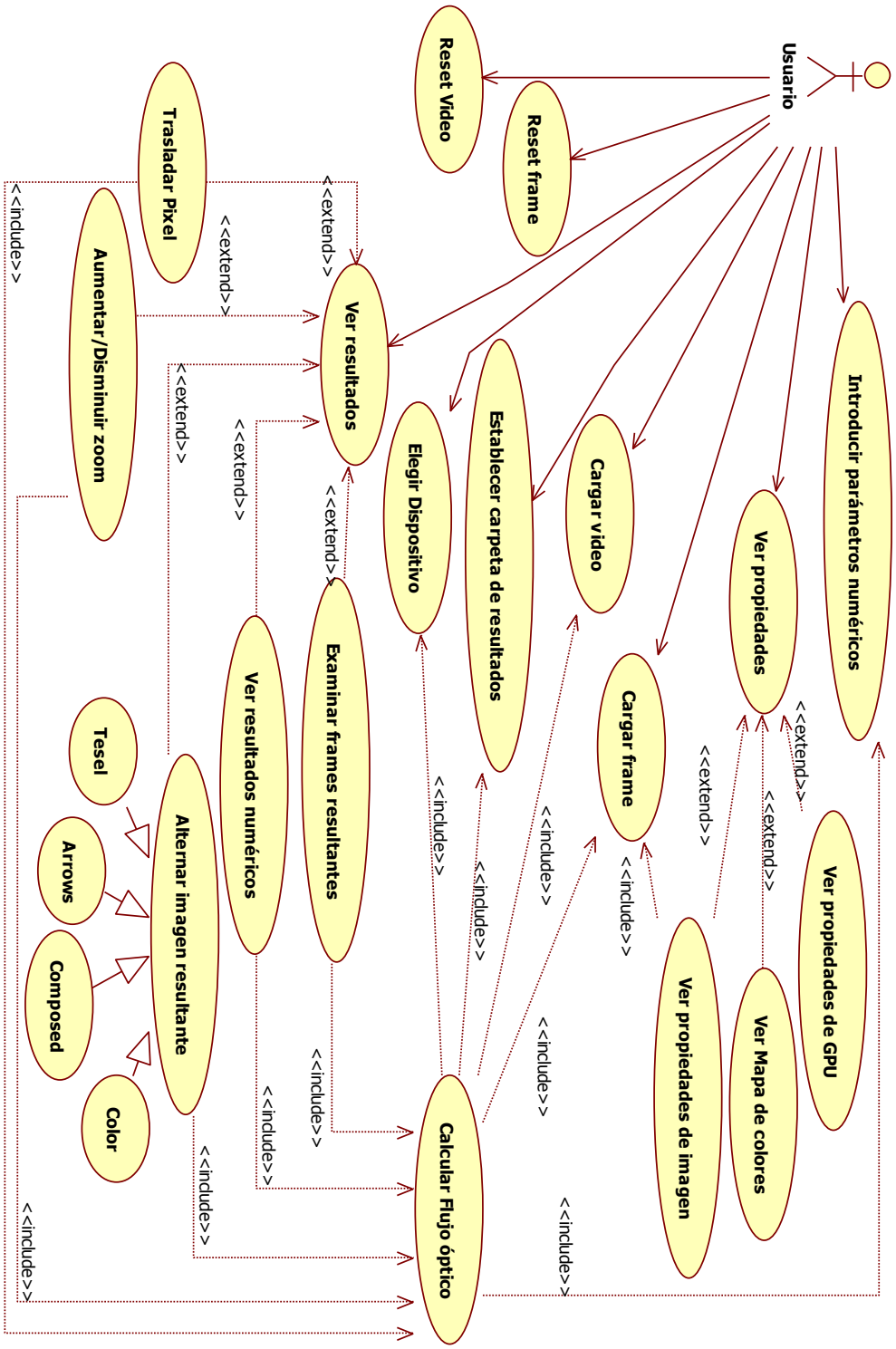


Imagen 4 - Modelos de casos de uso

Casos de uso detallados

Nombre	Cargar Frame	Identificador	CU-1
Actor principal		Usuario	
Descripción			
Al principio de la ejecución de la aplicación o en cualquier otro momento, pero siempre antes de realizar el cálculo del flujo óptico, el usuario deberá indicar la ruta de dos imágenes para que el algoritmo pueda seguir adelante. Para esto, se debe hacer click en el botón Load correspondiente y buscar en la ventana emergente el archivo de imagen que desea procesar seleccionándolo y haciendo click en aceptar.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El sistema muestra al usuario un diálogo de ficheros 2. El usuario navega por el sistema de archivos hacia la ubicación donde se encuentra la imagen 3. Selecciona la imagen de dos formas diferentes: haciendo doble click o uno solo y luego click en aceptar. 4. Se muestra la ruta en la etiqueta del nombre de la ventana principal			
Flujo Alternativo			
Excepción			
Si la ruta es inválida o el fichero no existe, no se carga ninguna imagen y hay que repetir el proceso			
Includes			
Requisitos especiales			
Notas			

Tabla 10- Caso de uso CU-1 Cargar Frame

Nombre	Cargar Video	Identificador	CU-2
Actor principal		Usuario	
Descripción			
Al igual que en el caso de uso CU-1, siempre antes de realizar el cálculo del flujo óptico, el usuario deberá indicar la ruta de la secuencia de video del cual extraer pares de frames y así realizar el cómputo. Para esto, se debe hacer click en el botón Load correspondiente y buscar en la ventana emergente el archivo de video que desea procesar seleccionándolo y haciendo click en aceptar.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El sistema muestra al usuario un diálogo de ficheros 2. El usuario navega por el sistema de archivos hacia la ubicación donde se encuentra el video 3. Selecciona el video de dos formas diferentes: haciendo doble click o uno solo y luego click en aceptar. 4. Se muestra la ruta en la etiqueta del nombre de la ventana principal			
Flujo Alternativo			
Excepción			
Si la ruta es inválida o el fichero no existe, no se carga ninguna imagen y hay que repetir el proceso			
Includes			
Requisitos especiales			
Notas			

Tabla 11 - Caso de uso CU-2 Cargar Video

Nombre	Reset Frame	Identificador	CU-3
Actor principal		Usuario	
Descripción			
Tras haber cargado un frame, el usuario puede tomar la decisión de borrarlo para cargar otro. Esta acción se lleva a cabo haciendo click en el botón de reset.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario hace click en el botón reset de la imagen correspondiente 2. Se elimina el nombre de la etiqueta de rutas			
Flujo Alternativo			
Excepción			
Si no hay ningún elemento que haya sido cargado previamente, no se lleva a cabo ninguna operación.			
Includes			
Requisitos especiales			
Notas			

Tabla 12 - Caso de uso CU-3 Reset frame

Nombre	Reset video	Identificador	CU-4
Actor principal		Usuario	
Descripción			
Tras haber cargado un video, el usuario puede tomar la decisión de borrarlo para cargar otro. Esta acción se lleva a cabo haciendo click en el botón de reset de la sección de video			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario hace click en el botón reset de video 2. Se elimina el nombre de la etiqueta de rutas			
Flujo Alternativo			
Excepción			
Si no hay ningún elemento que haya sido cargado previamente, no se lleva a cabo ninguna operación.			
Includes			
Requisitos especiales			
Notas			

Tabla 13 - Caso de uso CU-4 Reset Video

Nombre	Establecer carpeta de resultados	Identificador	CU-5
Actor principal		Usuario	
Descripción			
Por defecto, el sistema establece la carpeta personal como sitio donde guardar los frames resultantes de cada ejecución del flujo. No obstante, el usuario puede elegir donde almacenarlos.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario hace click en el botón de guardar resultados en una carpeta de destino 2. Se abre un dialogo de directorios 3. El usuario elige la carpeta deseada y hace click en aceptar 4. Se muestra la ruta hacia dicho directorio en la etiqueta correspondiente			
Flujo Alternativo			
Excepción			
Includes			
Requisitos especiales			
Notas			

Tabla 14 - Caso de uso CU-5 Establecer carpeta de resultados

Nombre	Introducir parámetros numéricos	Identificador	CU-6
Actor principal		Usuario	
Descripción			
La calidad de los resultados y la duración de la ejecución del proceso se basan, aparte del tamaño de las imágenes, en siete parámetros numéricos que son la función isotrópica "s" a través de la cual se obtiene lambda, alpha de la cual calculamos la constante de la ecuación, dt cuya inversa es tau, el número de iteraciones, la sigma cero y sigma N que determinan la bondad y cantidad de convolución gausiana con cada uno de los pares de frames y finalmente la tasa o rate que indica la velocidad a la que decrece sigma cero hasta llegar a sigma N. Por tanto, antes de proceder al cálculo en sí, el usuario deberá afinar estos parámetros según su criterio, o podrá dejar los que vienen por defecto.			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario introduce un valor numérico en cada una de las cuatro cajas 2. El valor se guarda automáticamente en la aplicación			
Flujo Alternativo			
Excepción			
En caso de que el usuario se equivoque introduciendo un carácter no numérico, la aplicación corregirá el error, volviendo a establecer el último valor válido.			
Includes			
Requisitos especiales			
Notas			

Tabla 15 - Caso de uso CU-6 Introducir parámetros numéricos

Nombre	Ver propiedades de GPU	Identificador	CU-7
Actor principal		Usuario	
Descripción			
A modo de información se permite al usuario poder evaluar las propiedades de la tarjeta gráfica con la que la aplicación realizará los cálculos en caso de que se seleccione la opción de CUDA. El usuario podrá ver estas propiedades desde un diálogo emergente que se encuentra en el menú principal			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario accede al menú View y despliega la lista de opciones 2. Elige la opción de "Grapihcs card properties"			
Flujo Alternativo			
1. El usuario utiliza el atajo de teclado, "g", que ejecuta directamente esta ventana emergente.			
Excepción			
En caso de que el hardware del equipo no presente tarjeta gráfica alguna, el diálogo se mostrará vacío.			
Includes			
Requisitos especiales			
Notas			

Tabla 16- Caso de Uso CU-7 Ver propiedades de GPU

Nombre	Ver propiedades de imagen	Identificador	CU-8
Actor principal		Usuario	
Descripción			
A modo de información se permite al usuario poder evaluar las propiedades de la/s imagen/es con la que la aplicación realizará los cálculos. El usuario podrá ver estas propiedades desde un diálogo emergente que se encuentra en el menú principal			
Trigger			
Precondición			
Haber cargado, al menos, un frame			
Postcondición			
Flujo Normal			
1. El usuario accede al menú View y despliega la lista de opciones 2. Elige la opción de “Image properties”			
Flujo Alternativo			
1. El usuario utiliza el atajo de teclado, “i”, que ejecuta directamente esta ventana emergente.			
Excepción			
Si el usuario ha reseteado ambas imágenes, el cuadro de resultados estará vacío.			
Includes		Incluye el caso de uso CU-1	
Requisitos especiales			
Notas			

Tabla 17 - Caso de uso CU-8 Ver propiedades de imagen

Nombre	Ver mapa de colores	Identificador	CU-9
Actor principal		Usuario	
Descripción			
A modo de información se permite al usuario poder ver en cualquier momento, la rueda de color según la cual se tomará como referencia para pintar los píxeles de las imágenes resultantes			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario accede al menú View y despliega la lista de opciones 2. Elige la opción de "Color wheel"			
Flujo Alternativo			
1. El usuario utiliza el atajo de teclado, "c", que ejecuta directamente esta ventana emergente			
Excepción			
Includes			
Requisitos especiales			
Notas			

Tabla 18 - CU-9 Ver mapa de colores

Nombre	Elegir dispositivo	Identificador	CU-10
Actor principal		Usuario	
Descripción			
El usuario elegirá, mediante un radio button, en qué procesador se ejecutará el algoritmo, es decir, si enteramente en la CPU o en la GPU también, dado que hay secciones que han de llevarse a cabo de forma secuencial			
Trigger			
Precondición			
Postcondición			
Flujo Normal			
1. El usuario hace click en uno de los dos botones de radio que hay antes de los contenedores de frames.			
Flujo Alternativo			
Excepción			
En caso de que el sistema sobre el que se esté ejecutando la aplicación no disponga de tarjeta gráfica que soporte CUDA o que no tenga instalados los drivers, el programa deshabilita automáticamente la opción de CUDA.			
Includes			
Requisitos especiales			
Notas			

Tabla 19 - Caso de uso CU-10 Elegir dispositivo

Nombre	Calcular flujo óptico	Identificador	CU-11
Actor principal		Usuario	
Descripción			
Desde que hay presentes en el sistema dos imágenes válidas, o un vídeo, el usuario puede hacer click en el botón "Compute" y el algoritmo realizará tantas iteraciones como se haya indicado en la entrada de datos, y al final guardará los resultados en el directorio establecido. Seguidamente mostrará en los 3 rectángulos inferiores el par de imágenes procesadas junto con su resultado			
Trigger			
Precondición			
El usuario tiene que haber cargado correctamente dos imágenes o un vídeo			
Postcondición			
Flujo Normal			
1. El usuario carga dos frames empleando el caso de uso CU-1 2. Seguidamente establece la carpeta donde desea que se almacenen las imágenes procesadas, como se indica en el caso de uso CU-5 3. Afina los parámetros numéricos del algoritmo 4. Elige si quiere una ejecución totalmente secuencial o paralela. 5. Comienza el proceso			
Flujo Alternativo			
1. El usuario carga un vídeo empleando el caso de uso CU-2 2. Seguidamente establece la carpeta donde desea que se guarden las imágenes tratadas, como se indica en el caso de uso CU-5 3. Regula los parámetros numéricos del algoritmo 4. Selecciona el tipo de procesamiento. 5. Comienza la ejecución.			
Excepción			
Si existe algún error en la configuración de los parámetros que han de establecerse previamente a la ejecución en si del algoritmo como por ejemplo el que sólo se haya cargado una imagen o ninguna, el programa mostrará un diálogo con una advertencia de error indicando que no se puede continuar hasta la correcta carga de todos los elementos necesarios. Por otra parte, si se cargan tanto dos frames como un vídeo, se procesarán todos los frames del video, por tener este prioridad.			
Includes		Este caso de uso incluye los casos de	

	uso CU-1, CU-2, CU-5, CU-6, CU-10
Requisitos especiales	
Notas	

Tabla 20 - Caso de uso CU-11 Calcular flujo óptico

Nombre	Examinar frames resultantes	Identificador	CU-12
Actor principal		Usuario	
Descripción			
Tras haber realizado el cálculo del flujo óptico con éxito, la aplicación guarda los resultados y los, muestra en la parte de abajo de la ventana principal. Aquí es donde se puede apreciar de forma gráfica lo preciso o impreciso que ha resultado la ejecución. Si se ha utilizado un video, se habilitan dos controles para el avance o retroceso para cada par de frames.			
Trigger			
Precondición			
Haber cargados los ficheros en la aplicación y haber calculado el flujo óptico.			
Postcondición			
Flujo Normal			
1. El usuario lleva a cabo los casos de uso CU-1, CU-5, CU-6, CU-10 y CU-11 2. Y Finalmente se cargan en los tres contenedores el par de frames originales junto con la imagen resultante ya sea compuesta o creada con la rueda de color			
Flujo Alternativo			
1. El usuario lleva a cabo los casos de uso CU-2, CU-5, CU-6, CU-10 y CU-11 2. Y Finalmente se cargan en el tercer contenedor, la secuencia de frames resultantes (compuesta o creada con la rueda de color) a la par que se habilitan los controladores anterior y siguiente para navegar por cada una que compone la secuencia			
Excepción			
Includes		Incluye el caso de uso CU-11	
Requisitos especiales			
Notas			

Tabla 21 - Caso de uso CU-12 Examinar frames resultantes

Nombre	Ver resultados numéricos	Identificador	CU-13
Actor principal		Usuario	
Descripción			
Tras haber realizado el cálculo del flujo óptico con éxito, la aplicación guarda los resultados numéricos del flujo óptico en un fichero binario a la par que los muestra en un diálogo en principio oculto al que puede accederse desde el menú principal de la aplicación.			
Trigger			
Precondición			
Haber cargados los ficheros en la aplicación y haber calculado el flujo óptico.			
Postcondición			
Flujo Normal			
1. El usuario lleva a cabo los casos de uso CU-1, CU-5, CU-6, CU-10 y CU-11 Seguidamente, el usuario accede al menú principal a través de View → Results o mediante el atajo de teclado R			
Flujo Alternativo			
2. El usuario lleva a cabo los casos de uso CU-2, CU-5, CU-6, CU-10 y CU-11 3. Seguidamente, el usuario accede al menú principal a través de View → Results o mediante el atajo de teclado R			
Excepción			
Includes		Incluye el caso de uso CU-11	
Requisitos especiales			
Notas			

Tabla 22 - Caso de uso CU-13 Ver resultados numéricos

Nombre	Alternar imagen resultante	Identificador	CU-14
Actor principal		Usuario	
Descripción			
Tras haber realizado el cálculo del flujo óptico con éxito, el usuario puede desear ver los resultados complementarios. Para esto tendrá que marcar cualquiera de los checkboxes que se encuentran deshabilitados para que la caja de imagen número tres muestre las combinaciones restantes resultantes del proceso. Estas pueden ser: 1. La imagen compuesta obtenida de la suma de la delta calculada por el algoritmo a la imagen inicial en lugar de la que se obtiene con la paleta de colores. 2. Esta opción muestra el frame número uno (el de la ventana izquierda) de fondo, a la vez que sobre él se pintarán los vectores de velocidad calculados anteriormente señalando estos el movimiento y la nueva orientación de los píxeles. 3. Tesel mostrará la imagen resultante de combinar pequeñas secciones rectangulares alternadas de ambos frames atendiendo a los píxeles originales y los trasladados respectivamente. Si el flujo está bien calculado, no se apreciará diferencia con el primer frame dado; no obstante, si no es el caso, podrán apreciarse franjas de distinto color en las áreas donde el flujo sea incorrecto por cuanto las nuevas coordenadas del pixel que se estará evaluando habrán sido incorrectas.			
Trigger			
Precondición			
Haber cargado correctamente los ficheros y haber realizado al menos una iteración del flujo óptico			
Postcondición			
Flujo Normal			
1. El usuario hace click en el checkbox Composed 2. Se carga la imagen compuesta en la caja de imagen número tres			
Flujo Alternativo			
1. El usuario hace click en Arrows 2. Se carga la imagen original en el tercer contenedor de la aplicación. 3. Se pintan los vectores de velocidad resultantes sobre las zonas de la imagen donde se ha detectado movimiento. Cuanto mayor la longitud, mayor es la velocidad.			

Excepción	
Si la caja de imágenes está vacía, no ocurrirá nada.	
Includes	Incluye el caso de uso CU-11
Requisitos especiales	
Notas	

Tabla 23 - Caso de uso CU-14 Alternar imagen resultante

Nombre	Aumentar/Disminuir zoom	Identificador	CU-15
Actor principal		Usuario	
Descripción			
Mientras el usuario está examinando los resultados generados por la aplicación la herramienta de zoom permite analizar con mayor facilidad la calidad y precisión de los mismos.			
Trigger			
Precondición			
Haber cargado correctamente los ficheros y haber realizado al menos una iteración del flujo óptico			
Postcondición			
Flujo Normal			
1. El usuario hace click en el botón con el símbolo (+) 2. La imagen resultante en el tercer cuadro al mismo tiempo que los vectores de velocidad (si estuvieren presentes) aumentará su tamaño en un 10%.			
Flujo Alternativo			
4. El usuario hace click en el botón con el símbolo (-) 5. La imagen resultante en el tercer cuadro al mismo tiempo que los vectores de velocidad (si estuvieren presentes) reducirá su tamaño en un 10%.			
Excepción			
Si la caja de imágenes está vacía, no ocurrirá nada.			
Includes		Incluye el caso de uso CU-11	
Requisitos especiales			
Notas			

Tabla 24 - Caso de uso CU-15 Alternar / Disminuir Zoom

Nombre	Trasladar Pixel	Identificador	CU-16
Actor principal		Usuario	
Descripción			
Es un método alternativo para verificar la bondad de los resultados en la cual seleccionando un pixel que se quiera verificar en el primer frame, se muestra en el segundo el mismo pixel dado que se le habrá aplicado el desplazamiento horizontal y vertical correspondientes.			
Trigger			
Precondición			
Haber cargado correctamente los ficheros y haber realizado al menos una iteración del flujo óptico			
Postcondición			
Flujo Normal			
1. El usuario hace click en un pixel cualquiera del primer contenedor de frames. 2. Automáticamente se señala, en rojo mediante una elipse, el pixel en su nueva posición en la segunda caja de imágenes.			
Flujo Alternativo			
Excepción			
Si la caja de imágenes está vacía, no ocurrirá nada.			
Includes		Incluye el caso de uso CU-11	
Requisitos especiales			
Notas			

Tabla 25 - Caso de uso CU-16 Trasladar pixel

Conclusiones

De esta forma, se puede establecer la estructura principal que debe poseer el programa:

- Menú principal con secciones de información general
- Disposición simple e intuitiva de los principales parámetros de configuración del algoritmo
- Diálogos de información independientes de la ventana principal con el fin de que puedan estar los dos en primer plano

Prototipos de validación

De forma paralela a la elaboración del proyecto, se ha ido realizando el diseño de prototipos de la aplicación. De esta manera, se ha gestionado la validación (actores y casos de uso) y las correcciones necesarias. Asimismo ha facilitado el identificar requisitos y a establecer objetivos.

Tras varias interacciones y modificaciones, los prototipos de validación son los siguientes:

Boceto 1 – Primera ventana de la aplicación

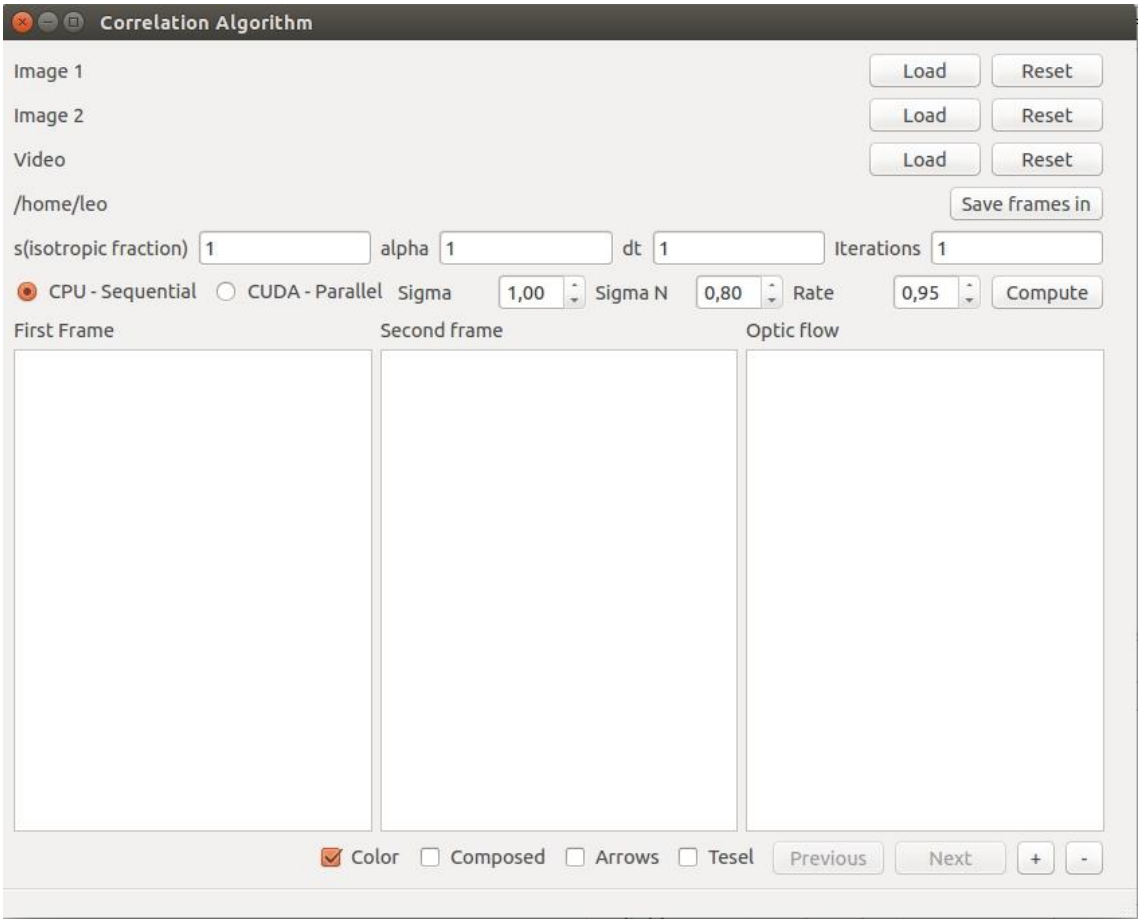


Imagen 5 - Segunda versión de la ventana principal

Boceto 2 – Cuadro informativo de las imágenes

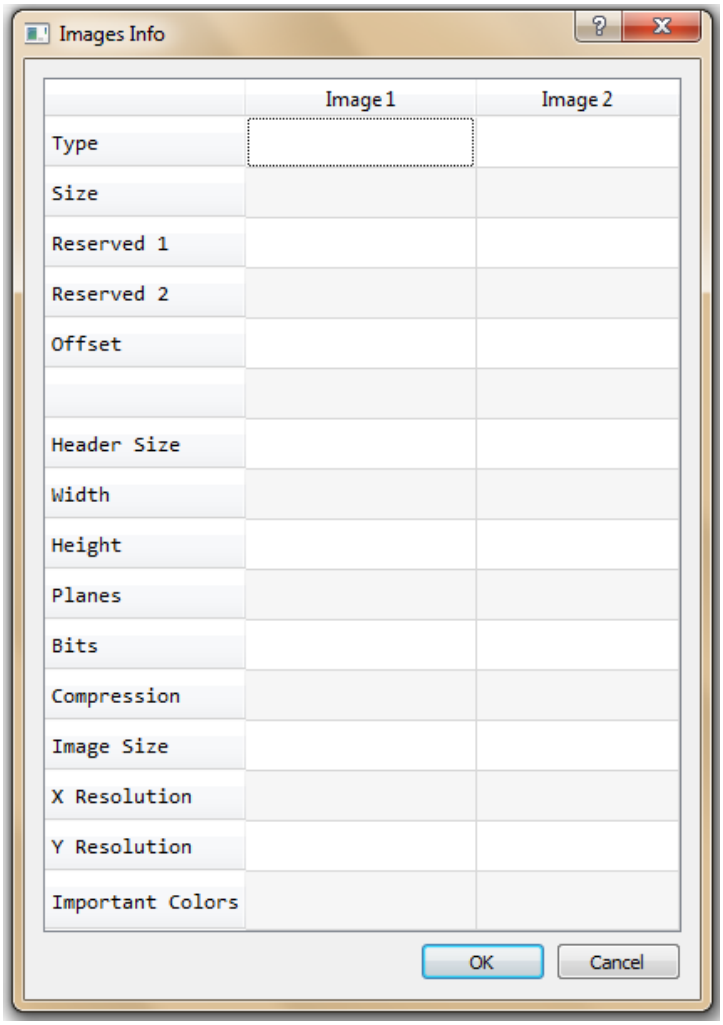


Imagen 6 - Boceto de diseño del cuadro de información de las imágenes

Boceto 3 – Cuadro de información de las GPUs

	Card 1	Description
Name	GeForce GT 540M	ASCII string identifying the device
totalGlobalMem	2147155968	total amount of global memory available on the device in bytes
sharedMemPerBlock	49152	maximum amount of shared memory available to a thread block in bytes; this amount is shared by all thread blocks simultaneo...
regsPerBlock	32768	maximum number of 32-bit registers available to a thread block; this number is shared by all thread blocks simultaneously resid...
warpSize	32	warp size in threads
memPitch	2147483647	maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through cudaMallocPitch()
maxThreadsPerBlock	1024	the maximum number of threads per block
maxThreadsDim[3]	1024, 1024, 64	contains the maximum size of each dimension of a block;
maxGridSize[3]	65535, 65535, 6...	contains the maximum size of each dimension of a grid;
clockRate	1344000	clock frequency in kilohertz;
totalConstMem	65536	total amount of constant memory available on the device in bytes;
major	2	major revision numbers defining the device's compute capability
minor	1	minor revision numbers defining the device's compute capability
textureAlignment	512	alignment requirement; texture base addresses that are aligned to textureAlignment bytes do not need an offset applied to text...
texturePitchAlignment	32	pitch alignment requirement for 2D texture references that are bound to pitched memory;
deviceOverlap	1	is 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not. Deprecated, use...
multiProcessorCount	2	the number of multiprocessors on the device;
kernelExecTimeoutEnabled	1	is 1 if there is a run time limit for kernels executed on the device, or 0 if not.
integrated	0	is 1 if the device is an integrated (motherboard) GPU and 0 if it is a discrete (card) component.
canMapHostMemory	1	is 1 if the device can map host memory into the CUDA address space for use with cudaHostAlloc()/cudaHostGetDevicePointer(),...
computeMode	0	is the compute mode that the device is currently in: Default, Exclusive, Prohibited, ExclusiveProcess
maxTexture1D	65536	is the maximum 1D texture size.

Cancel OK

Imagen 7 Cuadro de información de la GPU

Boceto 4 - Cuadro de estadísticas con los resultados de una ejecución

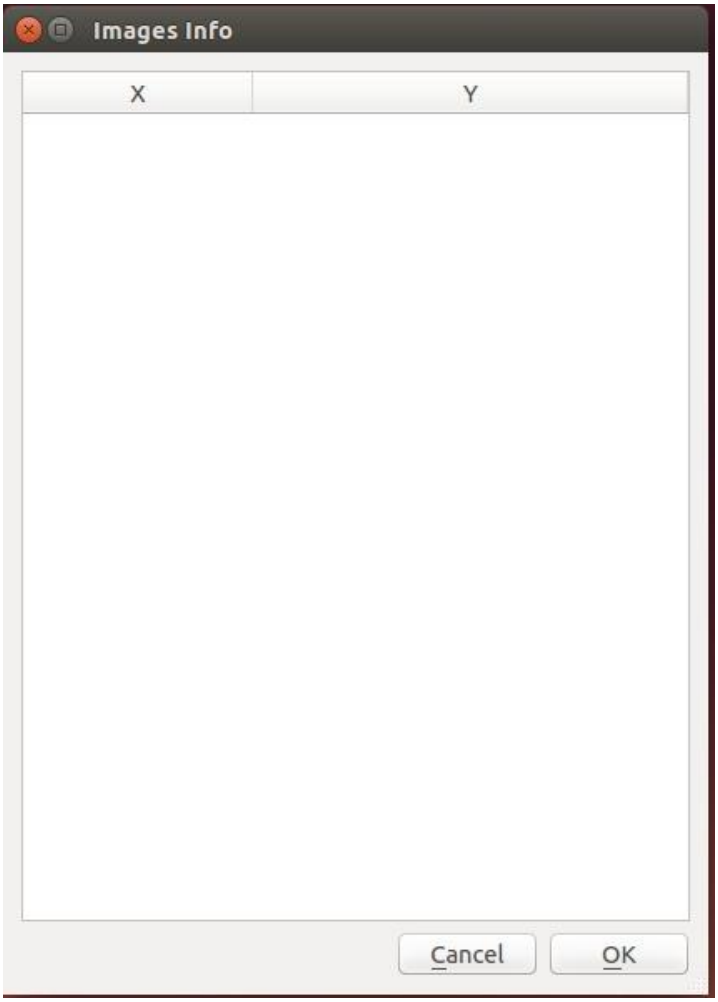


Imagen 8 - Cuadro de resultados por pixel

Diseño

Una vez obtenida toda la información en la fase de análisis, procederemos en el diseño a modelar la estrategia a seguir para la elaboración de la aplicación para que soporte todos los requisitos.

Introducción

En esta sección se explicará en profundidad la arquitectura general del software, representaciones de la interfaz y algoritmos.

Estructura

Para su elaboración se deberán tener en cuenta aspectos fundamentales tales como

- Facilidad de uso: el esfuerzo requerido para la utilización y/o aprendizaje del programa por parte del usuario ha de ser lo más pequeño posible
- Ser intuitivo.
- Robustez: el sistema ha de ser capaz de reponerse de posibles errores que puedan surgir tanto por parte del usuario como por posibles archivos corruptos, etc.
- Fiabilidad: ha de poder ejecutarse con normalidad y garantizar la operación libre de fallos durante su uso.
- Rendimiento: ha de gestionar un correcto uso de los recursos del sistema.
- Mantenibilidad: ha de permitir añadir y/o modificar los componentes existentes sin que se modifique el comportamiento del sistema
- Extensibilidad: ha de ser posible añadir nuevos componentes y capacidades al sistema sin que se vean afectados el resto de los componentes.

Teniendo en mente todo lo anterior se presenta a continuación un diseño genérico de la ubicación de cada una de las secciones necesarias para crear una aplicación útil, informativa y fácil de usar.

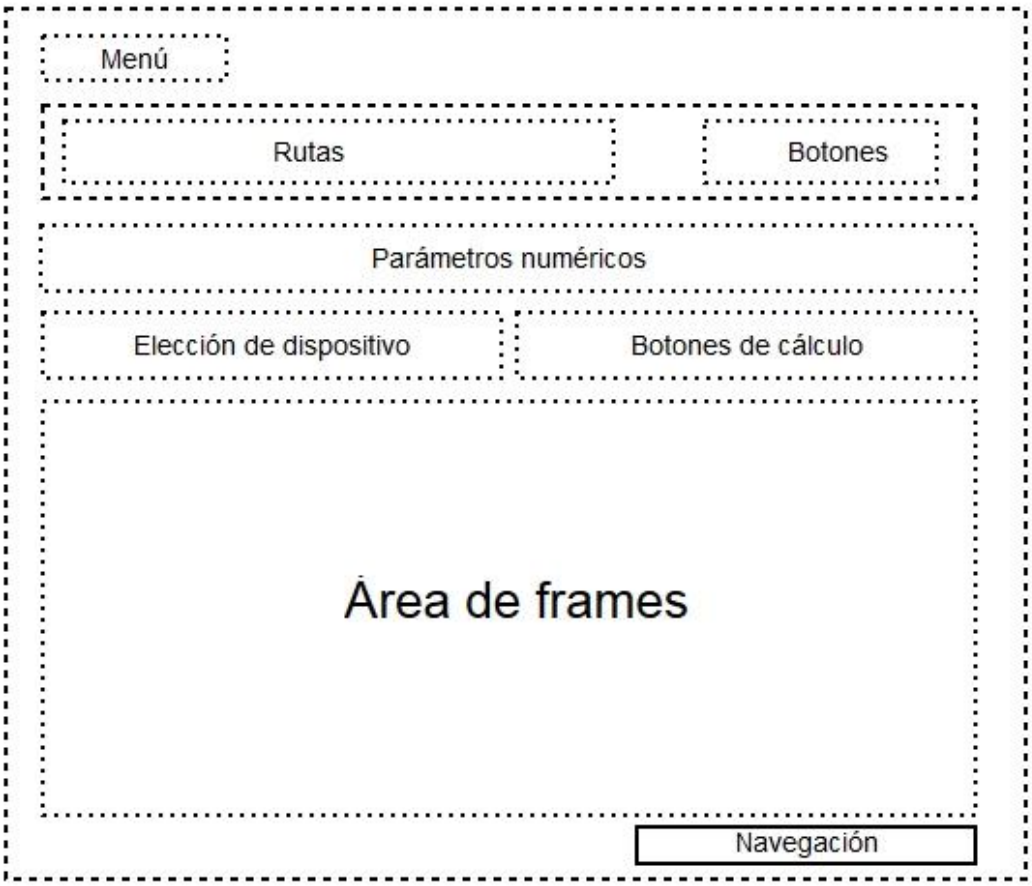


Imagen 9 - Estructura principal de la aplicación

Al acceder a cualquiera de las opciones del menú superior, se deberán desplegar cada uno de los apartados disponibles en la sección correspondiente.

Los botones pertenecientes a la segunda sección, funcionan en conjunción con las rutas. Cada botón de esa área permite cargar un archivo de imagen o un video tras lo cual, la ruta a dicho fichero se escribe en las etiquetas de la sección Rutas.

Los parámetros numéricos son cajas de entradas de tipo números reales con los que el algoritmo procesará las imágenes.

En la sección de elección de dispositivo, se podrá alternar entre la CPU o la GPU.

El botón de cálculo será el que inicie la ejecución del algoritmo, siempre y cuando la configuración sea correcta.

El área de frames junto con la navegación pertenece a la sección de resultados donde al término de todas las iteraciones del proceso, se mostrarán los resultados obtenidos, y se podrá avanzar o retroceder en caso de que lo que se haya procesado sea una secuencia de video.

Arquitectura

En cuanto al diseño arquitectónico, por simplicidad y por ser una aplicación que usará la tarjeta gráfica que presenta integrada en el sistema, se ha elegido una arquitectura monocapa en la que la capa de presentación, la de negocio y la de datos se encuentran en un solo nivel dado que el software se ha diseñado para ser ejecutado en un entorno de escritorio.

Patrón de diseño

Un patrón de diseño es una forma común de resolver un problema recurrente. Lo más importante no son las clases sino la forma en que están estructuradas y cómo funcionan juntas para resolver un problema dado de la mejor manera posible.

Para delinear el software de la aplicación se ha empleado un patrón Façade que es simple pero eficaz dado que oculta las complejidades del sistema y proporciona una interfaz al cliente mediante la cual acceder.

En el desarrollo de la aplicación se ha utilizado dicho patrón siguiendo el diagrama a continuación:

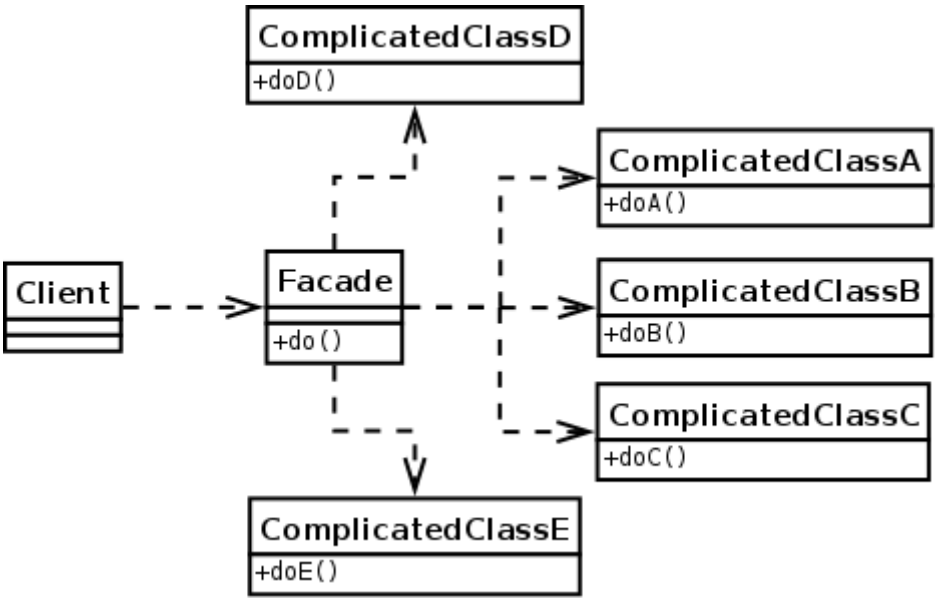


Imagen 10 - Patrón de diseño

De esta forma, será la clase "mainWindow", quien desempeñe el rol de fachada entre el cliente y la aplicación que, a su vez, es la encargada de gestionar todas las interacciones de la clase Imágenes y la clase VideoCapture (clase de OpenCV para manejar secuencias de video).

Diagrama de clases

A continuación se describe la estructura del sistema mostrando las clases del sistema y sus relaciones entre ellas.

Como se ha dicho en el apartado anterior, siguen el patrón Façade.

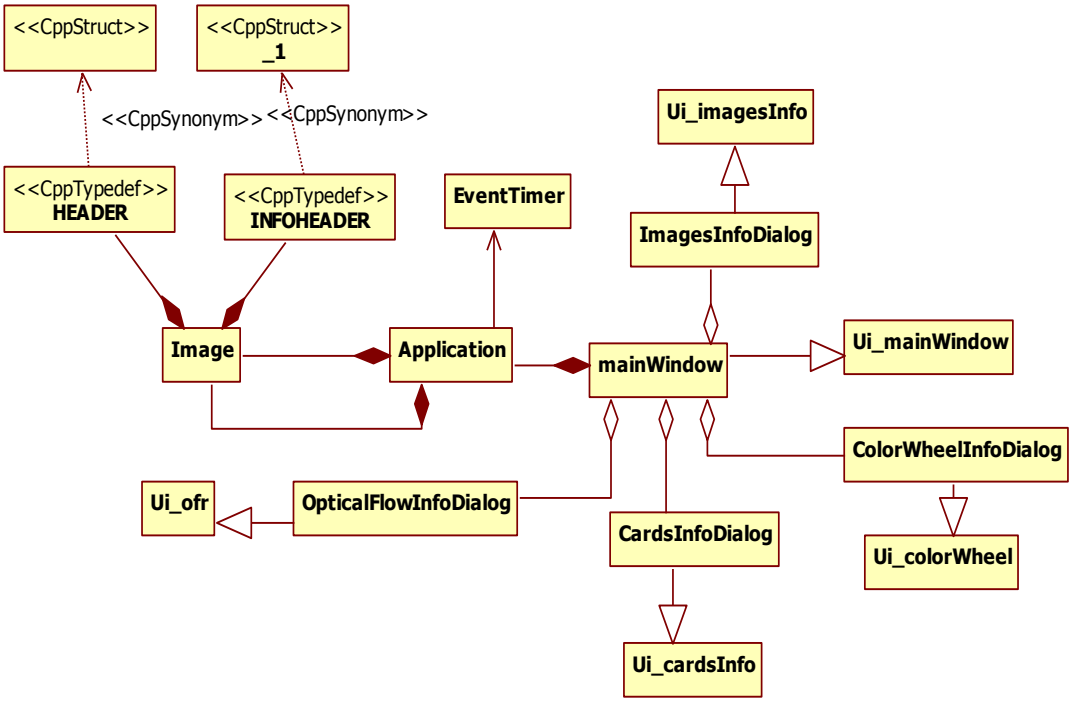


Imagen 11 - Diagrama de clases

Seguidamente se describen las tres clases más importantes: mainWindow, Application e Image.

Application

Se trata de la clase más importante del programa. Como su propio nombre indica, encapsula todos los atributos y métodos necesarios para el correcto proceso del algoritmo, gestionando las imágenes y/o video (frame a frame) y creando los sistemas de ecuaciones que luego han de ser resueltos para obtener el flujo óptico.

Sus atributos más importantes son las dos imágenes de las que se compone dado que son la base del algoritmo que se ha paralelizado en este proyecto.

Application

-I1: Image
-I2: Image
-videoFileName: QString
-resultsFolderName: QString
-lambdaVal: float
-dt: float
-cte: float
-nIter: int

<<create>>-Application(: QString, : QString, : float, : float, : float, : int)
<<create>>-Application(: QString, : float, : float, : float, : int)
<<create>>-Application()
<<destroy>>-Application()
+getImage1(: void): Image
+getImage2(: void): Image
+setResultsFolderName(: QString): void
+getResultsFolderName(: void): QString
+setIm1FileName(: QString): void
+getIm1FileName(: void): QString
+setIm2FileName(: QString): void
+getIm2FileName(: void): QString
+setVideoFileName(: QString): void
+getVideoFileName(: void): QString
+setLambda(: float): void
+getLambda(: void): float
+setCte(: float): void
+getCte(: void): float
+setDt(: float): void
+getDt(: void): float
+setIter(: int): void
+getIter(: void): int
+computeOpticFlow(: bool): int
-buildEquationSystem(: double, : double, : double, : double, : float, : int, : int, update: bool): void
-INDEX(i: int, j: int, N: int, M: int): int
-coeff(: double, : double, : double, : int, : int, : float, : float, : float, : float, : int, : int, update: bool): void
-smoothImagesAndNormalizeParams(: float, : float): void
-normalizeLambdaAndCte(: void): void
-runOpticFlow(: QString, : bool): void

Imagen 12 - Clase Application

Image

La segunda en importancia es la clase que se encarga del manejo y procesado de imágenes. Entre los métodos más importantes de esta clase podemos hallar las funciones responsables de obtener la derivada vertical y horizontal mediante la aplicación de kernels mediante la convolución de estos o la creación de imágenes nuevas a partir de un frame de video.

Sus atributos más importantes son nueve vectores, tres de los cuales son los canales rojo, verde y azul, uno es el que almacena la imagen convertida a

escala de grises, dos son sus correspondientes derivadas horizontal y vertical y los tres restantes representan las cuatro componentes de la matriz de correlación necesarias para calcular los coeficientes del sistema de ecuaciones que se explicará en el apartado siguiente.

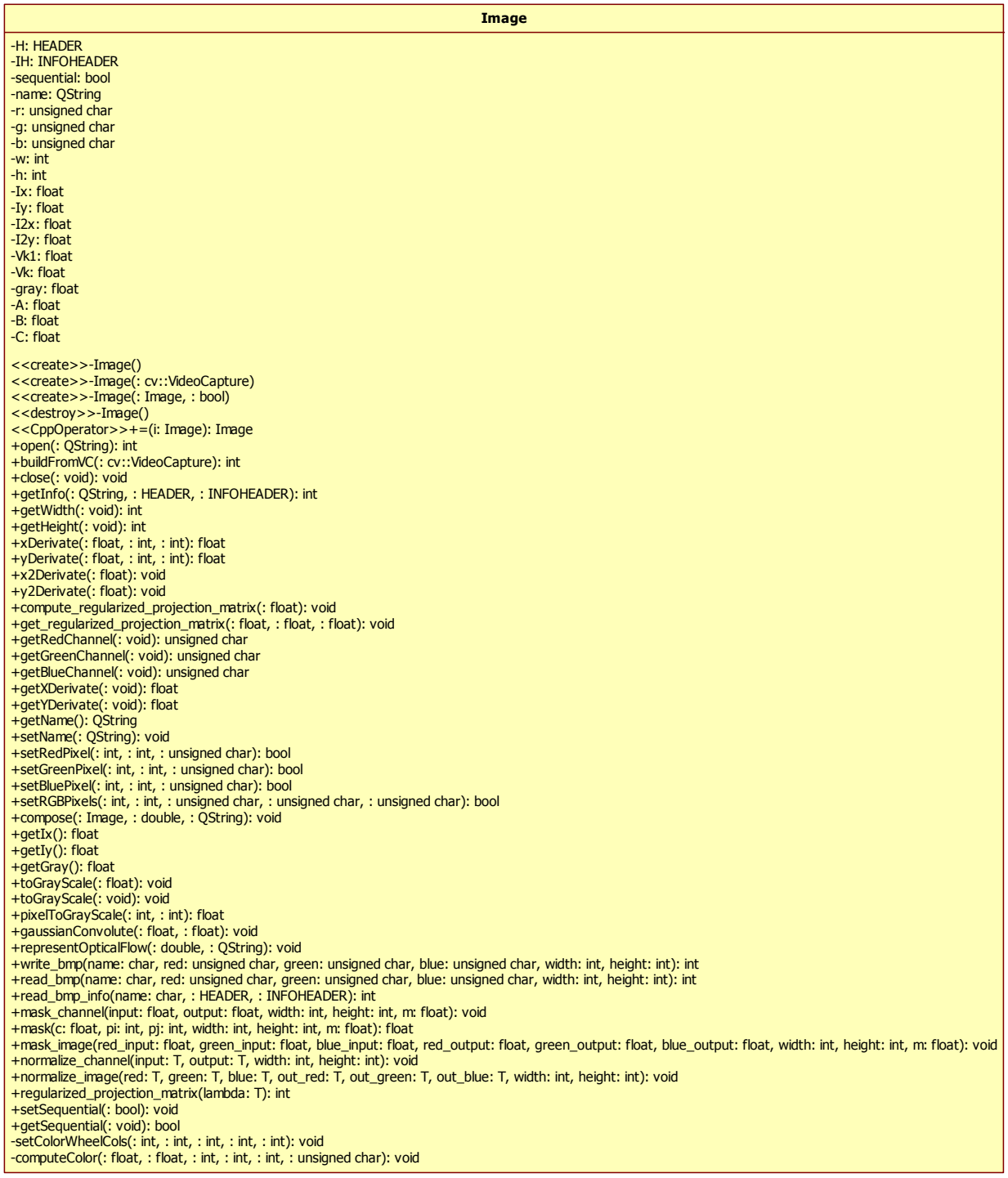


Imagen 13 - Clase Image

mainWindow

Finalmente, la ventana principal es quien implementa la fachada, entre la aplicación y el cliente o usuario final. Es decir, que es esta clase la responsable de la gestión de ficheros tanto como el mostrar los mensajes de error y la metainformación necesaria para la posible configuración del algoritmo.

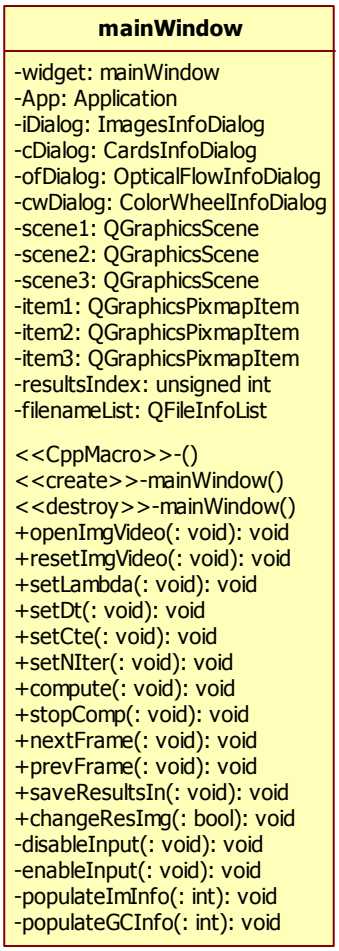


Imagen 14 - Clase mainWindow

Principio de responsabilidad individual

El motivo de que estas clases tengan gran número de métodos es por cuanto se ha puesto especial énfasis en que, tanto módulos como clases, sólo sean responsables de una sola parte de la funcionalidad proporcionada por el software, y que dicha responsabilidad esté encapsulada en su totalidad por la clase.

Diagramas de secuencia

Los diagramas de secuencia de una aplicación muestran la interacción de un conjunto de objetos a través del tiempo y se modelan para los casos de uso. El diagrama de secuencia contiene detalles del escenario, se indican los objetos y clases que realizan la comunicación y los mensajes que intercambian.

Para facilitar la compresión de los elementos, las tareas y los servicios que están presentes en el sistema, se muestran a continuación los diagramas de secuencia más relevantes y significativos de la aplicación.

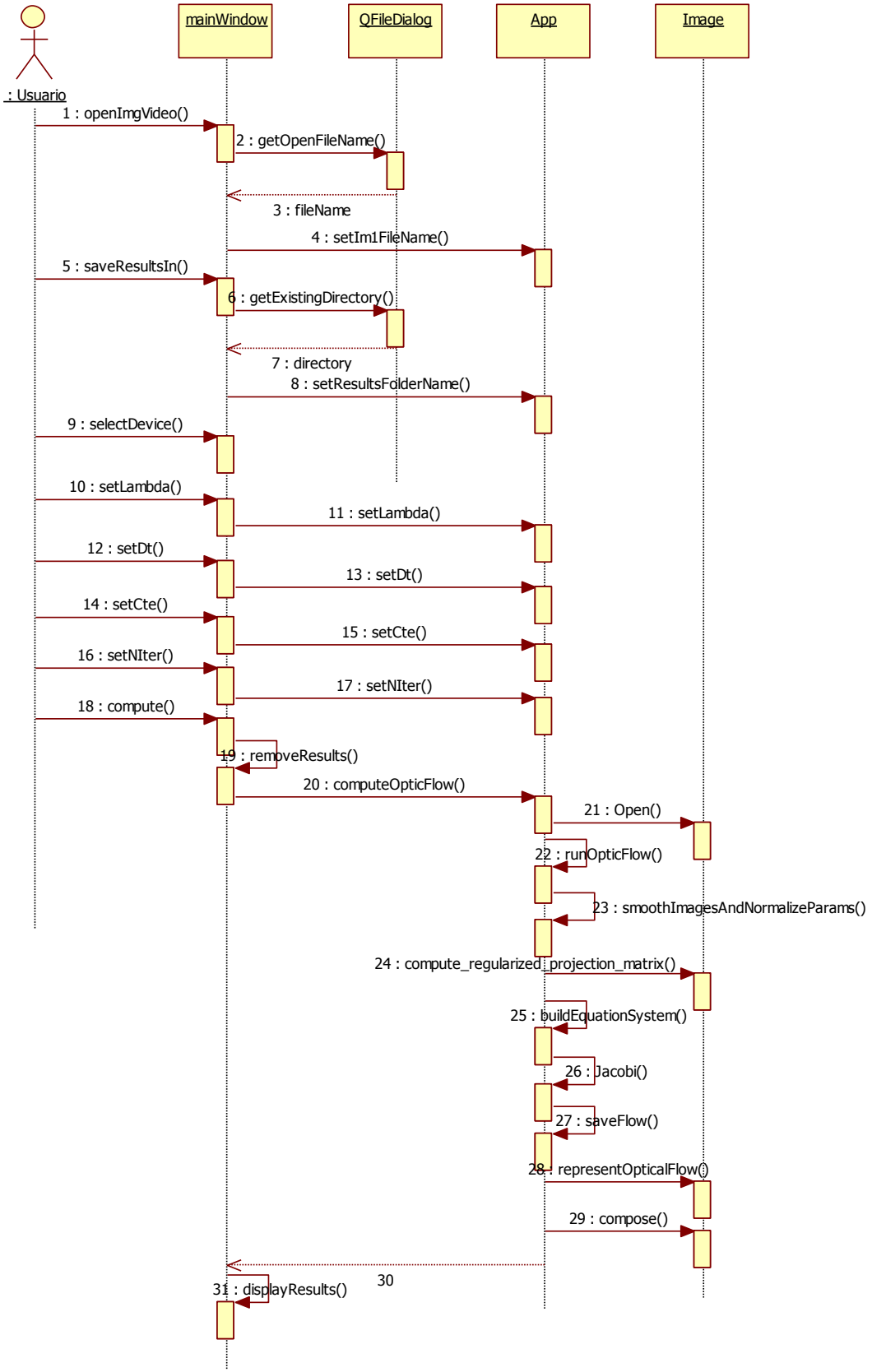


Imagen 15 - Diagrama de secuencia principal

El actor principal del diagrama de secuencia presentado interactuará a modo de interfaz con el elemento mainWindow, desarrollado en XML con el constructor de interfaces de QT, y traducido posteriormente a C++ como un fichero cabecera y un fichero de implementación. En concreto se trata de la interfaz principal del programa donde cada uno de los elementos posee un controlador dentro de la propia clase de mainWindow que es la encargada de actuar de intermediaria entre la aplicación (clase App) y el usuario.

A continuación se enumeran cada una de las llamadas y resultados presentes en el diagrama:

Apertura de ficheros

Un Usuario solicita la carga de dos imágenes y/o un video (1:openImgVideo). La ventana principal, mainWindow, solicita a la clase de diálogo de ficheros QFileDialog que presente una ventana a través de la que elegir el archivo mediante (2:getOpenFileName). Tras haberse elegido los elementos con los que se va a trabajar, estos son recibidos por la interfaz que se encargará de presentar el nombre de las imágenes en una etiqueta y por la aplicación (4: SetIm1FileName) para posteriormente cargarla en memoria.

Establecimiento de directorios

De la misma manera que en la apertura de ficheros, con la salvedad de que en esta ocasión la ruta que se obtiene es la de un directorio en lugar de la de un archivo, cuando se solicite establecer una carpeta para guardar los elementos resultantes (5:saveResultsIn) la ventana principal creará un diálogo emergente (6: getExistingDirectory) a través del cual el usuario elegirá un directorio que se almacenará en la clase aplicación (8: setResultsFolderName) a la par que se mostrará en la etiqueta correspondiente de la interfaz.

Configuración de parámetros

De entre los elementos que no dependen de un botón para cambiar su configuración, es decir, que se actualizan de forma dinámica, a medida que su contenedor de información cambia, el usuario podrá seleccionar uno a uno y modificar su valor (9:selectDevice), (10: setLambda), (12:setDt), (14:setCte) y

(16:setNIter). La vista enviará a continuación el valor nuevo al controlador de edición de la clase aplicación (11: setLambda), (13:setDt), (15:setCte) y (17:setNIter), tras haberlo procesado y comprobado que es correcto.

Cálculo del flujo óptico

A la hora de calcular el flujo óptico, el usuario solicita al programa que comience la ejecución por medio de un botón (18: compute). Internamente la ventana principal vaciará los resultados de cualquier ejecución previa (19: removeResults) dejándolo vacío y listo para asignar los nuevos. Es entonces cuando se indica a la aplicación que proceda a poner en marcha todas las acciones pertinentes (20:computeOpticalFlow) las cuales consisten en abrir las imágenes (21: Image::Open), configurarlas y comenzar a crear las variables necesarias para desarrollar la especificación del algoritmo (22: runOpticFlow) donde internamente se normalizan los parámetros introducidos por el usuario (23:smoothImagesAndNormalizeParams), se calcula la matriz de proyección (24: compute_regularized_projection_matrix) con la que se construye el sistema de ecuaciones (25: buildEquationSystem) y se resuelve con el método de jacobi (26:Jacobi). Finalmente, se guardan los resultados numéricos (27: saveFlow) e imágenes (28: representOpticalFlow), (29:compose). Cuando el control se retorna a la ventana principal esta los muestra en pantalla (31: displayResults).

Implementación

En este apartado se presentan cada uno de los aspectos de interés de la fase de desarrollo. El código fuente completo se adjunta a esta memoria en un CD. Además, la guía de uso se presenta en esta misma memoria en el Anexo 1 – Manual de usuario.

La implementación completa llevada a cabo a lo largo del proyecto de fin de carrera ha consistido en creación de una aplicación de escritorio con entorno gráfico de ventanas que se ejecute en un sistema operativo Linux que permita aunar sistemas tan aparentemente lejanos como lo son las aplicaciones gráficas y CUDA que es la tecnología que se ha empleado para paralelizar el algoritmo de flujo óptico.

El entorno de ventanas se ha basado en Qt.

La característica principal que debe poseer la aplicación es la de facilitar tanto la ejecución del algoritmo tanto como el análisis de resultados para así obtener una mayor y mejor visión de las virtudes y defectos del mismo.

Detalles de implementación

En este apartado se indican cada uno de los aspectos más relevantes de la codificación de la aplicación. Se presentan los detalles de mayor interés para facilitar la comprensión de los algoritmos y explicar el funcionamiento y estructura del sistema.

Estructura interna del sistema

Con el fin de agilizar el proceso de desarrollo se ha seguido la misma jerarquía y disposición interna para cada uno de los componentes. Todo ello basado en el patrón Façade, en el concepto de herencia y polimorfismo y en los diagramas de secuencia elaborados en la fase de diseño.

En concreto, tanto la ventana principal como cada uno de los diálogos de información (InfoDialog) son definidos y construidos gracias al soporte de las clases base *QMainWindow* y *QDialog* respectivamente. De este modo, la totalidad de las herramientas de la interfaz sigue un patrón sencillo y utiliza una misma metodología para la creación, modificación e interacción de sus elementos.

Debido a que todos los componentes gráficos mencionados siguen una disposición similar, se utilizará como ejemplo el listado de características de la tarjeta gráfica para explicar con más detalle el sistema realizado.

Listado de características

Cuando el usuario accede al menú "*Graphics card Data (G)*", se le presentará una ventana emergente que mostrará un cuadro de diálogo con una tabla que contendrá cada uno de los valores asociados a los atributos técnicos que la definen. Para acceder a esta ventana, deberá desplegar el menú *"View"* de la ventana principal y acceder a la opción mencionada al principio del párrafo. Mediante esta elección estará provocando que se ejecute la acción correspondiente, en este caso, "*triggered*", del correspondiente elemento de la lista (del menú "*View*") en la ventana principal que ha sido previamente conectada con el "*Slot*" (función que se ejecuta cuando se le emite una señal; explicado en el Apéndice. Detalles sobre la implementación) "*show*" perteneciente al cuadro de diálogo que se encargará de hacer visible la información.

En la siguiente imagen se apreciará de forma más clara de qué forma están conectados los elementos de la interfaz (mainwindow contiene a actionGraphicsData):

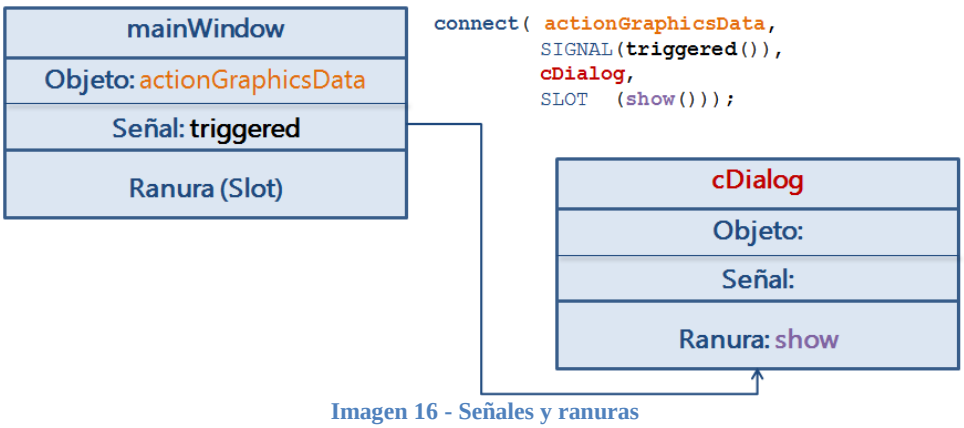


Imagen 16 - Señales y ranuras

Este funcionamiento es equivalente para el resto de ventanas de la aplicación por cuanto todas presentan el mismo comportamiento, incluso cuando a la información ha de aplicársele cierto formato antes de mostrarse. En la imagen anterior, se deduce que el diálogo aparecerá inmediatamente tras haberse desencadenado la señal dado que conecta con una función propia de la clase *QDialog* que no hace más que presentar el objeto en pantalla. No obstante, por ejemplo cuando se desean ver los resultados numéricos del flujo óptico, los números menores que 1 * 10^-6 se redondean a 0. En esta situación es posible definir un slot más complejo que permita procesar estos datos y finalmente invocar a la función show.

Edición de parámetros

Si en lugar de un listado de características, el usuario desea editar los parámetros numéricos que regulan el algoritmo como por ejemplo el número de iteraciones o elegir entre ejecución paralela o secuencial, la estructura interna y el comportamiento son similares a los explicados anteriormente. Cada botón, ya sea normal o de tipo radio (radiobutton), y cada caja de entrada de números o cada checkbox posee su conjunto particular de señales y ranuras las cuales se ejecutan dentro de la clase *mainwindow,* la cual a su vez, almacena dichos parámetros cada vez que cambian y los usa para configurar la aplicación.

Debido a la utilización del patrón Façade, la estructura interna del sistema ha sido implementada de forma equivalente para cada uno de los elementos anteriormente mencionados de tal forma que tenemos una sola interfaz capaz de comunicarse con los distintos subsistemas cada cual con sus respectivas funcionalidades (mainwindow, application, image, etc). De esta forma se ha agilizado el proceso de desarrollo y elaborado una arquitectura desacoplada, independiente y reutilizable.

Apertura de imágenes y vídeos

Otro aspecto muy importante de la estructura interna descrita, es la lectura inicial y su posterior carga en memoria de los frames que se han de procesar, ya sean dos individuales o los procedentes de un archivo de video. Para garantizar el correcto funcionamiento de ambos casos de uso, el desarrollo se ha basado en la

utilización de dos recursos existentes: la biblioteca *ami_bmp* [19] diseñado por el equipo de Luís Álvarez, profesor de la facultad de informática de Las Palmas de Gran Canaria y la clase *VideoCapture* de OpenCV [20] cuya función para este proyecto ha sido la de abrir el fichero de video indicado por el usuario y cargarlo en memoria para finalmente realizar una copia manual de cada componente RGB al vector interno perteneciente a la clase *Image* que se ha diseñado para poder encapsular la funcionalidad y gestionar el comportamiento asociado a las imágenes en el proyecto.

Según lo explicado hasta ahora, la secuencia normal es la siguiente:

1. Comunicación con la interfaz: el usuario hace click en el botón "*Load*" de la sección de imágenes, tras lo que se muestra en pantalla un diálogo de ficheros de tipo bmp, formato único que se puede manejar con la biblioteca ami_bmp (mainwindow.cpp) y posteriormente configura el resto de parámetros.

2. Ejecución de la ranura (slot): inmediatamente después producirse la señal asociada al click del usuario (signal), la función que controla la apertura del diálogo de ficheros procede a distinguir si el botón sobre el que se ha hecho click corresponde a una imagen (frame uno o frame dos) o al de video.

3. Selección del archivo: al conocer el tipo de fichero que se desea cargar, el algoritmo actúa en consecuencia y muestra una ventana de tamaño reducido donde filtra los formatos acorde al tipo anteriormente mencionado:

    a. Bmp, si es imagen
    b. Avi, mpg, mp4, flv o wmv, si es de un video

4. Almacenamiento de información: antes de finalizar su ejecución y posteriormente a la elección del usuario, el slot almacena el nombre y la ruta del archivo dentro de cada objeto imagen si el fichero es de tipo *bmp* o en una variable temporal perteneciente a la clase *application* para decidir el flujo de ejecución posteriormente a la hora de comenzar el cálculo del flujo óptico.

Es muy importante resaltar que, como medida de mejora y eficiencia, la imagen no se lee en esta etapa, es decir, se conoce su ubicación y se guarda su ruta para más tarde, cuando el usuario decida comenzar el proceso de cálculo, lo cual implícitamente confirma que el proceso de elección está libre de cualquier posible error humano, es entonces cuando se leerán y cargará el contenido de todos y cada uno de los píxeles en memoria y se procederá a obtener el flujo óptico.

5. Presentación al usuario: No obstante, si bien la imagen no se carga como tal en la clase *Image* sí que se muestra su miniatura en su correspondiente caja de imagen situada en la parte inferior de la ventana con el fin de aportar retroalimentación al usuario informándole de los resultados de sus operaciones.

Funcionalidad principal

El propósito principal de esta aplicación es obtener un par de resultados numéricos (horizontal y vertical) reales para cara pixel presente en la imagen (flujo óptico denso). El desempeño tiene lugar a partir del instante en que el usuario efectúa un click en el botón que dice "*Compute*" en la ventana principal. Es en ese momento cuando se comienza por leer las cabeceras de la imagen (se omite este paso si fuese un archivo de video dado que se infiere que todos los frames serán de las mismas dimensiones), tras lo cual se comprueba que estén ambas cargadas (o el video en su defecto) y, si son ficheros de tipo imagen, que presenten igual ancho e igual alto. En caso de que cualquiera de que estas comprobaciones previas falle, se mostrará automáticamente un cuadro emergente de notificación (QMessageBox) con el siguiente mensaje:

There are no Images nor video files loaded. You must load 2 images (with same dimensions) and/or a video file.

que indica que han de cargarse dos imágenes o un video, lo que hará reflexionar al usuario de cual haya podido ser el error producido.

Dado que se pueden realizar distintas ejecuciones empleando el mismo par de frames y teniendo en cuenta que los resultados tanto numéricos como gráficos

se guardan en un directorio dentro del equipo donde se ejecuta el programa, lo primero que hace el algoritmo es detectar si, en la carpeta actual elegida para guardar dichos resultados, existe algún rastro de los mismos y si los hay, procede a eliminarlos para dejar sitio a los de la última ejecución que se realiza inmediatamente a dicho borrado.

Cálculo del flujo óptico

Para poder explicar con más detalle el funcionamiento del cálculo, se aborda su explicación paso a paso:

1. Inicialización: el primer paso consiste en dirimir qué tipo de archivo se va a procesar. Si se produce el caso de que se han cargado tanto dos imágenes individuales a la par que una secuencia de video, el controlador dará prioridad a esta última.

2. Apertura o construcción de imágenes: posterior a la distinción del tipo de fichero, se procederá a la apertura de los mismos con la correspondiente comprobación de errores (mismas dimensiones, apertura exitosa de ambas imágenes o secuencia de video, etc.). Si se produjese cualquier error, el controlador cerrará cualquier archivo abierto, devolverá cero como resultado indicando así que no se ha procesado ningún pixel y por ende la ventana principal no mostrará ningún resultado.
   En el caso particular de un video, se procederá además a la importación del contenido de los píxeles de cada par de frames a cada uno de los dos objetos de tipo Image que están asociados a la clase Application, pudiendo de esta forma, continuar con la ejecución normal del programa.

3. Establecimiento del modo: se indica al algoritmo si las funciones que ha de utilizar son las de la CPU (secuenciales) o las de CUDA (paralelas). Para esto se utiliza el parámetro inyectado en la función que proviene de la ventana principal donde ha sido indicado por el usuario.

4. Suavizado: siguiendo el modelo del documento, la imagen se convierte a escala de grises simplificando así su tratamiento y además se le aplica un desenfoque gaussiano con el objetivo de suavizar y difuminar sus bordes

de tal forma que tras sucesivos usos los desplazamientos largos se reduzcan.

5. Creación del sistema de ecuaciones: Seguidamente se colocan los coeficientes en el vector que hará las veces de matriz escasa, su vector de filas, su vector de punteros de columnas y el término independiente. Todos estos términos se explican más abajo. Lo importante es que utilizando esta estructura se construye el sistema descrito por las ecuaciones 17 y 18 (previamente despejadas) del documento del proyecto.

6. Resolución: Finalmente el sistema de ecuaciones se resuelve mediante el método de Jacobi, el cual podrá iterar hasta el tope estipulado. Cabe destacar, como nota al margen, que se ha empleado este método debido a que la naturaleza simultánea de la actualización de las soluciones hacen de Jacobi un procedimiento inherentemente paralelizable mientras que, en principio Gauss-Seidel es secuencial.

Evaluación de resultados

Una característica fundamental de la aplicación es que permite alternar entre los distintos tipos de resultados construidos derivados del flujo óptico calculado para cada pixel durante la fase más importante de la ejecución. La Imagen 9 muestra en su parte inferior la sección del área de navegación donde, una vez finalizada la etapa anterior, se puede alternar entre los diversos resultados compuestos. A continuación se muestra cada una de las variaciones extraídas a partir de la imagen de los cuadrados negros sobre fondo blanco, los cuales se mueven en distintas direcciones y distintas longitudes:

1. Composed

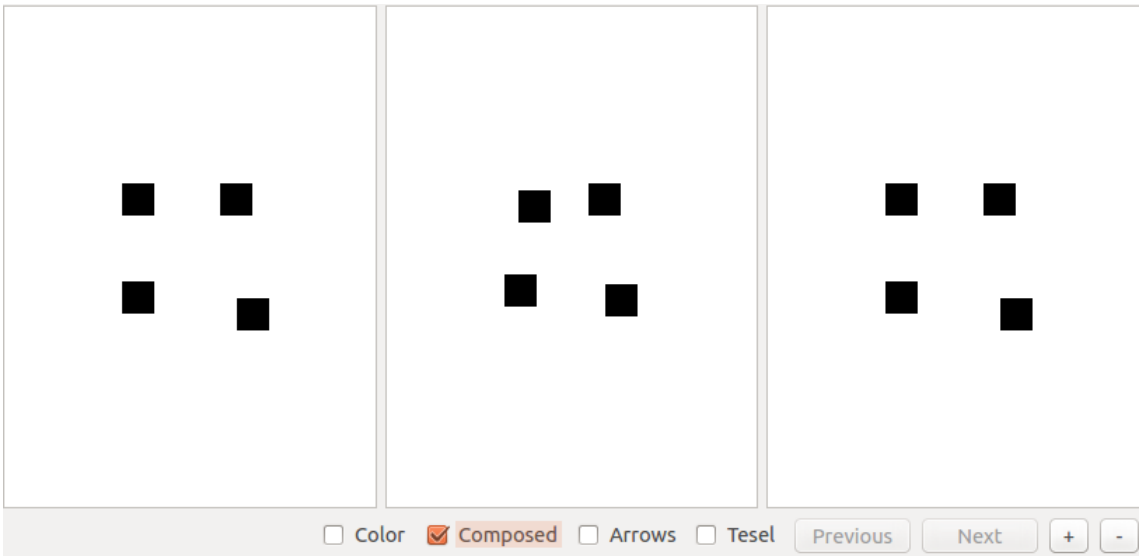


Imagen 17 - Resultado compuesto

2. Color

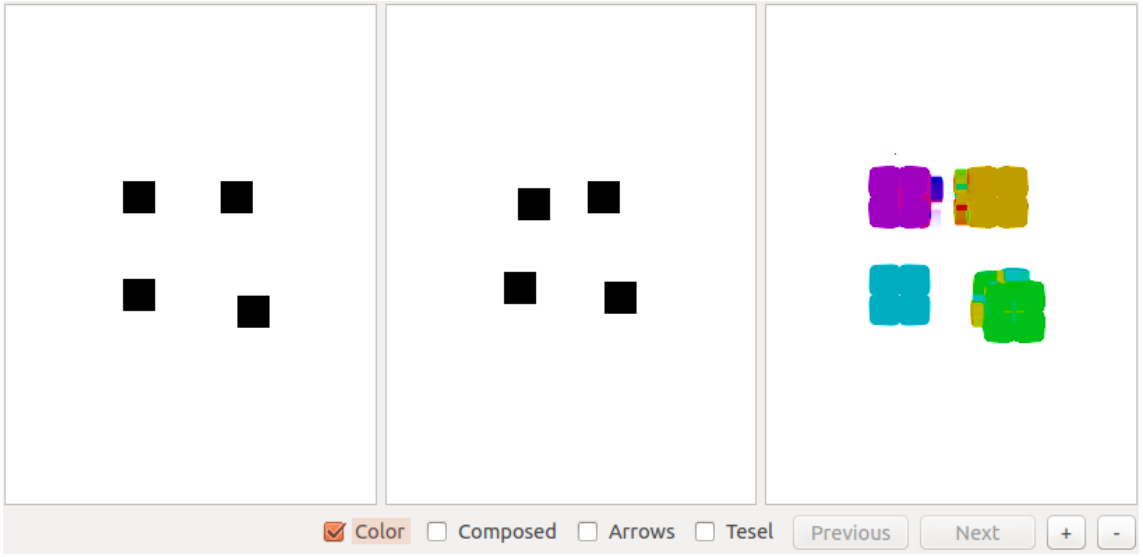


Imagen 18 - Resultado según rueda de color

3. Arrows

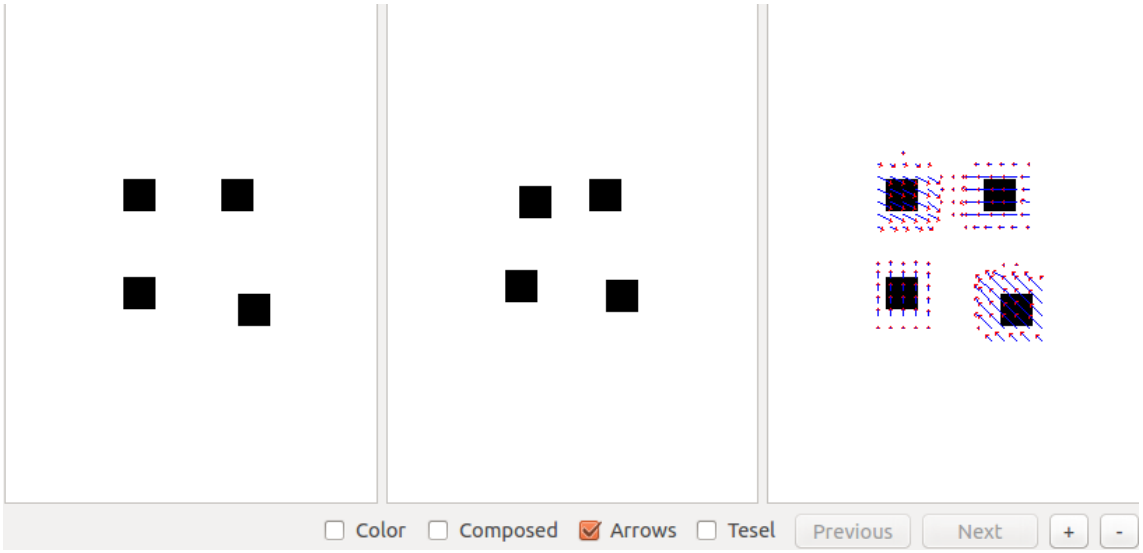


Imagen 19 - Representación de vectores de movimiento

4. Tesel

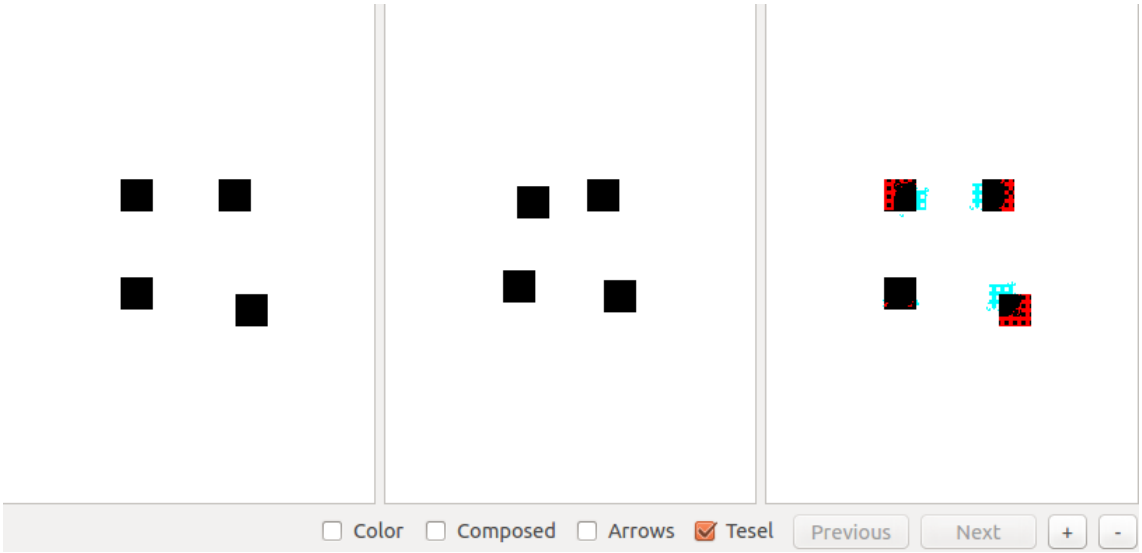


Imagen 20 - Tesel

5. Interactiva

   Nota: la flecha roja no sale en los resultados.

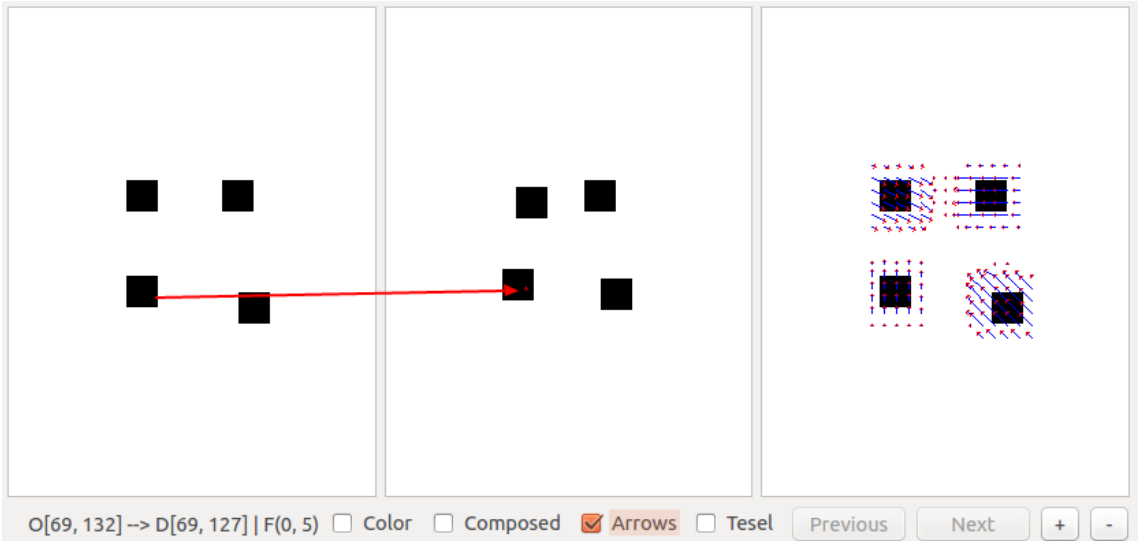


Imagen 21 - Modo interactivo

Para poder realizar este filtrado de imágenes, de tal forma que sólo se vea un resultado a la vez, cada uno de los diferentes tipos de imágenes se almacena en disco con diferentes extensiones:

1. .ofc.bmp. *ofc* quiere decir *optical flow composed*. Significa que el contenido de la imagen (de tipo "*bmp*") combina píxeles tanto del primer frame como del segundo. Se entiende que es más correcta cuanto más idéntica es al primer frame elegido dado que surge de combinar pequeños fragmentos de este junto a los trasladados (a los que se les ha aplicado el desplazamiento obtenido) en la segunda imagen.

2. .ofr.bmp. *ofr* quiere decir *optical flow result*. En esta sección se podrá observar una imagen donde las diferentes regiones de la imagen están pintadas de diferentes colores siguiendo el modelo descrito en la paleta de referencia de tal manera que los pixeles que se desplacen hacia arriba tendrán un color como la parte superior de la imagen de referencia y de igual manera ocurre con los demás extremos.

3. .tes.bmp. *tes* representa la imagen teselada (*tesel*) que se constituye de la misma forma que la "*ofc*" pero por subregiones de pixeles o mini ventanas en lugar de píxel a píxel.

4. .vsq.bmp. Este formato no es más que una copia del frame original por cuanto se emplea para representar los vectores de movimiento. En otras palabras, se pinta primeramente una copia de la primera imagen de la secuencia y sobre ella se indica el movimiento mediante flechas de tronco azul y punta roja.
   Por otra parte, para no sobrecargar la visión con demasiadas flechas de forma que los resultados sean comprensibles, el método adoptado ha sido subdividir la imagen en regiones de menor tamaño y calcular la media horizontal y la vertical pintando así el origen en el pixel inicial y el punto final del vector en el señalado por los promedios.

5. Como método interactivo, se ha brindado la posibilidad de que el usuario se capaz de seleccionar un píxel aleatorio dentro del área definida por la primera caja de imagen (mediante un click de ratón) y automáticamente sus coordenadas se trasladarán previa suma de las componentes horizontal y vertical y se representará dicho pixel en la caja de imagen contigua usando una elipse de color rojo alrededor delimitando las nuevas coordenadas.

Además, cuando se elige un archivo de video, al final del proceso habrá dentro de la carpeta de resultados un número de imágenes igual al total de frames que componen dicho video menos uno (dado que el flujo se calcula por pares). Para navegar y observar la bondad de cada uno, en la sección de navegación se habilitarán dos botones: "*Previous*" y "*Next*" que permitirán iterar, como si de un bucle se tratase, a lo largo de cada uno de los ítems de la carpeta.

Finalmente, teniendo en cuenta que la calidad de los resultados se evalúan a nivel de pixeles y conjugando este factor con que, por comodidad y facilidad de uso, es necesario mostrar además el par de imágenes que han dado lugar al contenido que se observa en la tercera caja de imagen (image box) de la ventana principal, se han añadido por tanto dos botones, de zoom, que permiten ensanchar o disminuir el área del tercer frame construido con el objetivo de apreciar con mayor precisión la bondad del mismo.

Paralelización

Integración

Gestionar y reducir la complejidad en programas de gran envergadura hace necesario dividirlos en componentes responsables de porciones pequeñas y bien definidas. La compilación independiente es un aspecto muy importante de C y C++, lo que permite que las diferentes partes de un programa sean traducidas en objetos separados y luego enlazadas entre sí para formar un archivo ejecutable. El desarrollo de programas que empleen la GPU, por muy grandes y complejos que resulten, no difiere en ningún aspecto.

En la imagen a continuación puede observarse un ejemplo reducido del proceso de compilación de la aplicación según la estructura con la que ha sido diseñada. El objetivo principal, aparte de mantener la abstracción y poder reutilizar al máximo todas las clases a la vez que poder ser capaces de mantener aisladas unas unidades de otras durante el proceso de desarrollo, ha sido poder integrar y compilar dentro de un mismo proyecto, código C/C++ (incluyendo las características especiales de Qt) junto con código CUDA que ha sido diseñado para ejecutarse en los núcleos de las GPUs y que además presenta una sintaxis distinta en ciertas secciones de la codificación.

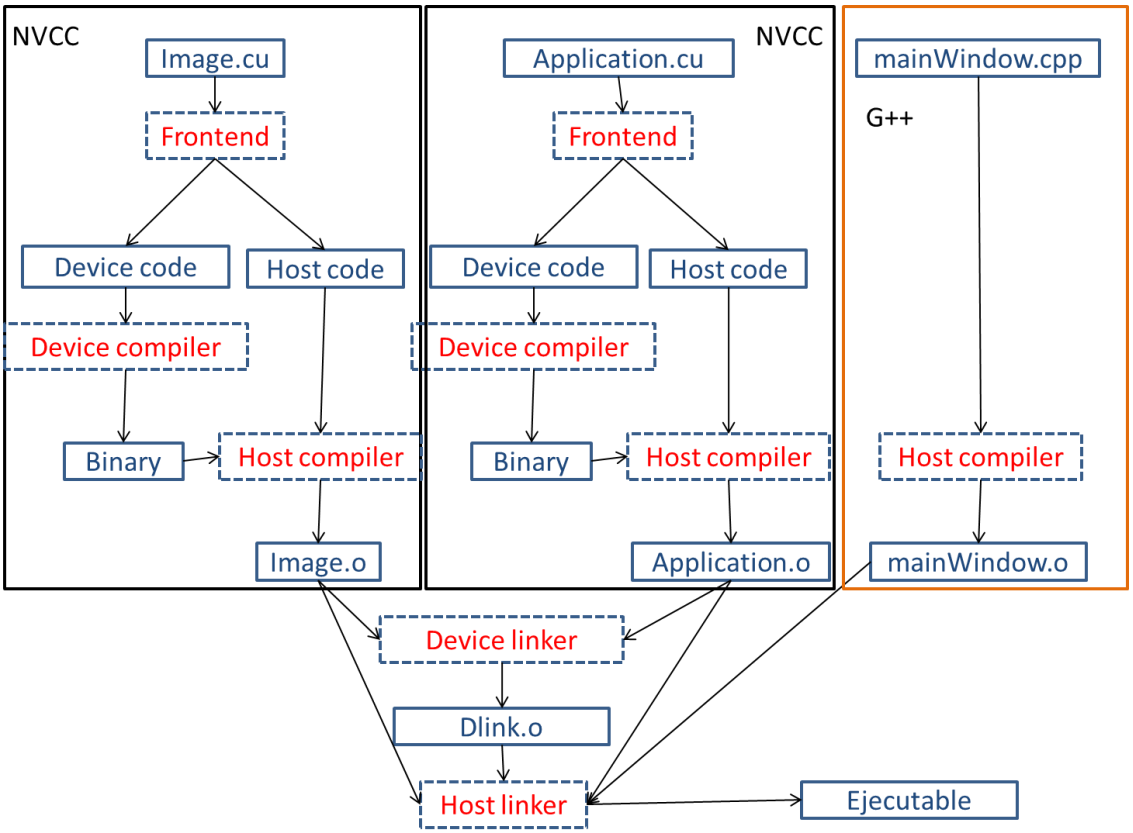


Imagen 22 - Proceso de compilación

En la ilustración se aprecia de manera simple y reducida (sólo se muestran tres ficheros fuente) las etapas de las que consta el proceso de compilación, enlazado y generación de un fichero ejecutable por parte de las diferentes herramientas a partir de los ficheros C/C++ y CUDA.

De forma muy breve, el método para lograr este fin consiste en compilar todos los ficheros fuente CUDA a ficheros objeto (.o) para posteriormente enlazarlos con los ficheros objeto resultantes de haber sido compilados mediante G++. Finalmente todos estos archivos se enlazan mediante G++.

El aspecto clave para el éxito de la operación es la palabra reservada *extern*. Dado que la compilación del código CUDA ha de realizarse por separado, es necesario incluir en las unidades de C++ el prototipo de cada una de las funciones necesarias, es decir, de todas las funciones invocadas de tal forma que en la etapa de enlazado se resuelvan todas las referencias a dicha función y la generación del ejecutable se acabe con éxito.

Ocupación de la GPU

Un aspecto importante en la toma de decisiones durante la fase de programación ha sido elegir el tamaño de bloque antes de comenzar a ejecutar un núcleo (kernel). Es importante entender las restricciones de cada kernel y la GPU en la que se está ejecutando para poder seleccionar un tamaño de bloque que dará como resultado un buen rendimiento. Una heurística común utilizada para elegir un buen tamaño de bloque es apuntar a una alta ocupación, que es la relación entre el número de warps (conjunto de 32 hilos o threads; ver diccionario de conceptos) activos por multiprocesador y el número máximo de warps que pueden estar activas en el multiprocesador a la vez. Una mayor ocupación no siempre significa un rendimiento más alto, pero es una métrica útil para medir la capacidad de ocultación de la latencia de un núcleo.

Realizar los ajustes pertinentes para calcular la ocupación, sobre todo en las versiones antiguas de CUDA es difícil. Implica implementar una serie de cálculos complejos que deben tener en cuenta la GPU del equipo sobre el que se está ejecutando el software y sus capacidades (incluyendo los registros y el tamaño de memoria compartida), y las propiedades del kernel (uso de memoria compartida, registros por hilo, hilos por bloque).

No obstante, a día de hoy, la API de CUDA provee bastantes funciones que brindan soporte para estos cálculos en tiempo de ejecución. Encontramos, por ejemplo, que la función *cudaOccupancyMaxActiveBlocksPerMultiprocessor* efectúa una predicción basada en el tamaño de bloque y el uso de memoria compartida de un kernel e informa la ocupación en términos del número de bloques de hilos simultáneos por multiprocesador.

Por otra parte, también hayamos entre dichas funciones aquellas que permiten configurar el lanzamiento de hilos basándose en la ocupación *cudaOccupancyMaxPotentialBlockSize* y *cudaOccupancyMaxPotentialBlockSizeVariableSMem*. Ambas calculan de forma heurística el tamaño de bloque que logra la máxima ocupación.

Seguidamente se muestra un breve fragmento teórico de 11 líneas de código en la que se puede ver la filosofía detrás de estas funciones y entender así mucho más rápido de qué manera se obtiene, en tiempo de ejecución, el mejor tamaño de bloque con el que conseguir la máxima ocupación de la GPU.

```
int dimBloque;    // Configurador de lanzamiento  - Tamaño de bloque
int minDimGrid;   // Mínimo tamaño de grid (conjunto de bloques)
                  // necesario para lograr la máxima ocupación para un
                  // lanzamiento completo del dispositivo
int dimGrid;      // Tamaño de grid real necesario
cudaOccupancyMaxPotentialBlockSize( &minDimGrid, &dimBloque, kernel,0,
                                                                    0);
// Redondeo al alza según el tamaño del vector
dimGrid = (dimVector + dimBloque - 1) / dimBloque;

kernel<<< dimGrid, dimBloque >>>(vector, dimVector);
```

De esta forma se ha evitado tener que rehacer los cálculos en caso de que deba ejecutarse el algoritmo en otra máquina a la vez que se ha mejorado en la portabilidad del mismo.

Texturas

Las texturas son otra variedad de memoria de sólo lectura que pueden mejorar el rendimiento y reducir el tráfico de memoria cuando las lecturas presentan ciertos patrones de acceso. Al residir en la cache del chip, a menudo proporciona un mayor ancho de banda por cuanto los accesos a memoria RAM se verán reducidos. Estas memorias están específicamente diseñadas para patrones de acceso que muestran gran localidad espacial, es decir, que mejorará las latencias cuando los hilos lean posiciones de memoria contiguas o cercanas a las que otros hilos ya han accedido como es el caso de este proyecto donde además de tener cuenta el pixel actual (x, y) se han de considerar también sus 8 vecinos.

En nuestro caso particular el kernel que ha sido mayormente beneficiado es el encargado de realizar cada iteración del algoritmo de Jacobi que sufre de bastantes problemas a consecuencia del formato de matriz elegido (ver apartado Matriz escasa). El más importante radica en la manera en la que los hilos dentro de un warp (conjunto de hilos que permite acceso coalesente a memoria) acceden a los índices y a los arrays de datos. Mientras que, para una fila en concreto, los índices de columna y los valores (distintos de cero) se almacenan de forma contigua, estos valores no se acceden simultáneamente de forma que aumenta la latencia.

Una alternativa al núclceo escalar (kernel), es el núcleo vectorial, el cual asigna un warp a cada fila de la matriz. Este, a diferencia del anterior que utiliza un

hilo por cada fila de la matriz, requiere coordinación entre los hilos correspondientes al mismo warp. Específicamente se requiere una reducción paralela para sumar los resultados parciales de cada hilo. Su ventaja principal es el tener acceso a índices y datos contiguos.

No obstante ha de tenerse en cuenta que la flexibilidad adicional que brinda este formato de almacenamiento (CSR) por cuanto permite un número variable de elementos por fila, introduce también un nivel de complejidad extra: la divergencia de hilos la cual ocurre cuando se producen saltos condicionales en el código. Por ejemplo, cuando el núcleo se aplica a una matriz con un número muy variable de elementos por fila, es probable que muchos hilos dentro de un warp permanezcan inactivos mientras que el hilo con la fila más larga continúe iterando. Dado que los warps se ejecutan de forma independiente, esta forma de divergencia de hilos es menos pronunciada en el núcleo vectorial.

Sin embargo, a pesar de que hasta ahora todo parece indicar que la segunda versión del kernel encargado de realizar el producto entre la matriz y el vector será más rápida y eficiente, en nuestro caso particular ocurre precisamente al revés porque la ejecución eficiente del núcleo vectorial exige que las filas de la matriz contengan un número distintos de cero mayor que el tamaño del warp (32) y dado que nuestra matriz se emplea con el objetivo de representar las ecuaciones para cada pixel donde se contabilizan como máximo sus ocho vecinos y la componente vertical, tenemos que el mayor número de elementos no nulos por fila será de 10. Como resultado, el rendimiento del núcleo vectorial es sensible al tamaño de la fila de la matriz y ostensiblemente menor con respecto a su versión escalar. De hecho en los experimentos, duraba como mímimo el doble de tiempo.

Precisión simple versus doble

Al principio de la implementación, con el fin de obtener resultados mas fiables y precisos se emplearon cifras de doble precisión (double). No obstante, si bien los resultados finales eran correctos, la ejecución resultaba como máximo un 40% más lenta en comparación con la CPU. Tras una investigación en profundidad se descubrió que tanto en general como en con la del PC de desarrollo, las tarjetas gráficas de Nvidia, históricamente presentan un peor

rendimiento cuando se opera en precisión doble en comparación con la precisión simple. De hecho, esto puede comprobarse con la tabla resumen de la arquitectura Fermi (diseño que sigue la GPU de nuestro PC) que se observa a continuación [21] en la que se aprecia que el número de operaciones en coma flotante de precisión doble es la mitad que las de precisión simple. Esto es debido a que alberga mayor cantidad de FPU32 (floating processing units de 32 bits) que FPU64.

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

Imagen 23 - Arquitectura Fermi

Por este motivo, se cambió la definición de cada una de las funciones y miembros de las clases para que, simpre que fuesen números reales, usen el tipo *float* en lugar de *double*. Sólo por este cambio se consiguió que la ejecución en paralelo fuese más veloz que la secuencial, incluso sin el uso de cache ni memoria compartida.

Métricas

En este apartado se explica brevemente las dos formas de medir la productividad del algoritmo paralelizado a la vez que por qué se ha elegido usar los eventos de CUDA.

Medición mediante CPU

Para ilustrar este método se muestra a continuación un breve fragmento de código en el que se puede apreciar la forma de medir el tiempo de forma secuencial:

```
cudaMemcpy(d_a, a, N * sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, N * sizeof(float), cudaMemcpyHostToDevice);

elapsed = CPUTimer();
kernel<<<gridSize, blockSize>>>(d_a, d_b, N);
cudaDeviceSynchronize();
elapsed = CPUTimer() - elapsed;
```

Además de las dos llamadas a *CPUTimer*(), se emplea también la barrera de sincronización explícita *cudaDeviceSynchronize*() para bloquear la ejecución de la CPU hasta que se hayan completado todos los comandos emitidos anteriormente en el dispositivo. Sin esta barrera, este código mediría el tiempo de lanzamiento del kernel y no su tiempo de ejecución.

Medición mediante eventos CUDA

El problema del método anterior es que se bloquea el pipeline de la GPU. Por esta razón, CUDA ofrece una alternativa a través de su API de eventos, la cual incluye llamadas para crear, destruir y registrar eventos y, a su vez, calcular el tiempo transcurrido en milisegundos entre dos eventos cualesquiera registrados. Seguidamente se muestra un ejemplo:

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaMemcpy(d_a, a, N * sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, N * sizeof(float), cudaMemcpyHostToDevice);

cudaEventRecord(start);
kernel<<<gridSize, blocksize>>>(N, 2.0f, d_x, d_y);
cudaEventRecord(stop);

cudaEventSynchronize(stop);
float elapsed = 0;
cudaEventElapsedTime(&elapsed, start, stop);
```

Los eventos de CUDA hacen uso del concepto de los flujos de CUDA que son simplemente una secuencia de operaciones realizadas en orden en el

dispositivo. Las operaciones en diferentes flujos pueden intercalarse y, en algunos casos, superponerse.

En el fragmento anterior, *cudaEventRecord*() hace que el dispositivo registre una marca de tiempo para el evento cuando llegue a ese punto en la secuencia. La función *cudaEventSynchronize*() bloquea la ejecución de la CPU hasta que se graba el evento especificado, mientras que *cudaEventElapsedTime*() devuelve en el primer argumento el número de milisegundos de tiempo transcurrido entre la grabación de inicio y fin.

Ancho de banda

El ancho de banda de memoria también se calcula con estos medios. Para evaluar la eficiencia de ancho de banda, se usa tanto el ancho de banda máximo teórico como el efectivo.

Ancho de banda teórico

El ancho de banda teórico se puede calcular utilizando las especificaciones de hardware del producto. Por ejemplo, la GPU NVIDIA Tesla M2050 utiliza RAM DDR (double data rate, o, doble tasa de transmisión de datos) con una velocidad de reloj de memoria de 1.546 MHz y una interfaz de memoria de 384 bits de ancho. Conociendo estos elementos de datos, el ancho de banda de memoria teórica máxima de la NVIDIA Tesla M2050 es de 148 GB / seg, como se calcula a continuación:

$$Ancho\ de\ banda_{\ teórico} = 1546 * 10^6 * \left(\frac{384}{8}\right) * \frac{2}{10^9(GB/s)} = 148{,}416\ GB/s$$

Ancho de banda efectivo

Este se calcula mediante la sincronización de las actividades específicas del programa y por saber cómo nuestro programa accede a los datos:

$$Ancho\ de\ banda_{\ efectivo} = \left(\frac{Lb + Eb}{t}\right) * \frac{1}{10^9}$$

Donde Lb y Eb son el número de bytes leídos y escritos por kernel repectivamente, y t es el tiempo transcurrido en segundos.

Rendimiento

Otra métrica muy importante de funcionamiento es el rendimiento, el cual se mide en GFLOP/s:

$$GFLOP/s_{\ efectivo} = \left(\frac{2 * N}{t * 10^9}\right)$$

N es el número de elementos del kernel, y t es el tiempo transcurrido en segundos. Al igual que el ancho de banda máximo teórico, el GFLOP / s puede deducirse de las especificaciones del producto (pero calcularlo puede ser un poco complicado porque es muy dependiente de la arquitectura). Por ejemplo, la GPU Tesla M2050 tiene un rendimiento teórico de coma flotante en precisión simple de 1030 GFLOP / s, y un rendimiento teórico de doble precisión de 515 GFLOP / s.

Construcción del modelo

El problema de flujo óptico en realidad se reduce a la resolución de un sistema lineal grande de ecuaciones de la forma:

$$A\,x = b$$

donde A es la matriz de coeficientes cuadrada, definida positiva y diagonal dominante de tamaño $n\,x\,n$. Por su parte, x y b son vectores de n elementos los cuales representan el flujo óptico (horizontal y verical) en el instante t y $t-1$ respectivamente.

Para trasladar el modelo matemático presentado en el documento (sistema de ecuaciones) a C++, primeramente es menester despejar las dos ecuaciones que conforman el sistema de tal forma que los términos independientes queden agrupados a un lado y los dependientes al otro.

Resolución de la ecuación

Primeramente se comenzará con la ecuación del movimiento horizontal.

$$\frac{U_{i,j}^{k+1} - U_{i,j}^{k}}{\tau} = C\left(\frac{a_{i+1,j} + a_{i,j}}{2}\frac{U_{i+1,j}^{k+1} - U_{i,j}^{k+1}}{h_1^2} + \frac{a_{i-1,j} + a_{i,j}}{2}\frac{U_{i-1,j}^{k+1} - U_{i,j}^{k+1}}{h_1^2}\right.$$
$$+ \frac{c_{i,j+1} + c_{i,j}}{2}\frac{U_{i,j+1}^{k+1} - U_{i,j}^{k+1}}{h_2^2} + \frac{c_{i,j-1} + c_{i,j}}{2}\frac{U_{i,j-1}^{k+1} - U_{i,j}^{k+1}}{h_2^2}$$
$$+ \frac{b_{i+1,j+1} + b_{i,j}}{2}\frac{U_{i+1,j+1}^{k+1} - U_{i,j}^{k+1}}{2h_1h_2}$$
$$+ \frac{b_{i-1,j-1} + b_{i,j}}{2}\frac{U_{i-1,j-1}^{k+1} - U_{i,j}^{k+1}}{2h_1h_2}$$
$$- \frac{b_{i+1,j-1} + b_{i,j}}{2}\frac{U_{i+1,j-1}^{k+1} - U_{i,j}^{k+1}}{2h_1h_2}$$
$$\left. - \frac{b_{i-1,j+1} + b_{i,j}}{2}\frac{U_{i-1,j+1}^{k+1} - U_{i,j}^{k+1}}{2h_1h_2}\right)$$

$$+ I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) * \left(I_1\left(\bar{x}_{i,j}\right) - I_2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) + U_{i,j}^{k} * I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) + V_{i,j}^{k} * I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)\right) - U_{i,j}^{k+1} * I_{2x}^2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) - V_{i,j}^{k+1} * I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) * I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$

Para facilitar el proceso, renombramos las variables:

$$A_1 = \frac{a_{i+1,j} + a_{i,j}}{2h_1^2} \qquad A_2 = \frac{a_{i-1,j} + a_{i,j}}{2h_1^2}$$
$$B_1 = \frac{b_{i+1,j+1} + b_{i,j}}{4h_1h_2} \qquad B_2 = \frac{b_{i-1,j-1} + b_{i,j}}{4h_1h_2}$$
$$B_3 = \frac{b_{i+1,j-1} + b_{i,j}}{4h_1h_2} \qquad B_4 = \frac{b_{i-1,j+1} + b_{i,j}}{4h_1h_2}$$
$$C_1 = \frac{c_{i,j+1} + c_{i,j}}{2h_2^2} \qquad C_2 = \frac{c_{i,j-1} + c_{i,j}}{2h_2^2}$$

$$U_1 = U_{i+1,j}^{k+1} - U_{i,j}^{k+1} \qquad U_2 = U_{i-1,j}^{k+1} - U_{i,j}^{k+1}$$

$$U_3 = U_{i,j+1}^{k+1} - U_{i,j}^{k+1} \qquad U_4 = U_{i,j-1}^{k+1} - U_{i,j}^{k+1}$$

$$U_5 = U_{i+1,j+1}^{k+1} - U_{i,j}^{k+1} \qquad U_6 = U_{i-1,j-1}^{k+1} - U_{i,j}^{k+1}$$

$$U_7 = U_{i+1,j-1}^{k+1} - U_{i,j}^{k+1} \qquad U_8 = U_{i-1,j+1}^{k+1} - U_{i,j}^{k+1}$$

Reescribiendo queda por tanto:

$$\frac{U_{i,j}^{k+1} - U_{i,j}^{k}}{\tau} = \mathrm{C}(A_1 * U_1 + A_2 * U_2 + C_1 * U_3 + C_2 * U_4 + B_1 * U_5 + B_2 * U_6 - B_3 * U_7 - B_4 * U_8) + I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)\left(I_1\left(\bar{x}_{i,j}\right) - I_2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) + U_{i,j}^{k} * I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) + V_{i,j}^{k} * I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)\right) - U_{i,j}^{k+1} * I_{2x}^2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) - V_{i,j}^{k+1} * I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) * I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$

Renombrando otra vez:

$$I_1 = I_1\left(\bar{x}_{i,j}\right) \qquad I_2 = I_2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$
$$I_{2x} = I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) \qquad I_{2y} = I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$

Queda:

$$\frac{U_{i,j}^{k+1} - U_{i,j}^{k}}{\tau} = \overbrace{\mathrm{C}(\ldots)} + \underbrace{I_{2x}\left(I_1 - I_2 + U_{i,j}^{k} * I_{2x} + V_{i,j}^{k} * I_{2y}\right)}_{} - \overbrace{U_{i,j}^{k+1} * I_{2x}^2 - V_{i,j}^{k+1} * I_{2y} * I_{2x}}$$

Los miembros del lado derecho de la ecuación que tiene la llave por encima son términos de índice k+1. Los demás son términos k.

El siguiente paso consiste en despejar los términos k+1, a la izquierda:

$$\frac{U_{i,j}^{k+1}}{\tau} - \mathrm{C}(\dots) + U_{i,j}^{k+1} * I_{2x}^2 + V_{i,j}^{k+1} * I_{2y} * I_{2x} = U_{i,j}^{k}\left(I_{2x}^2 + \frac{1}{\tau}\right) + I_{2x}\left(I_1 - I_2 + V_{i,j}^{k} * I_{2y}\right)$$

Reescribimos de forma extendida el término izquierdo para sacar factor común de $U_{i,j}^{k+1}$

$$\frac{U_{i,j}^{k+1}}{\tau} - \mathrm{C}\Big(A_1\left(U_{i+1,j}^{k+1} - U_{i,j}^{k+1}\right) + A_2\left(U_{i-1,j}^{k+1} - U_{i,j}^{k+1}\right) + C_1\left(U_{i,j+1}^{k+1} - U_{i,j}^{k+1}\right) + C_2\left(U_{i,j-1}^{k+1} - U_{i,j}^{k+1}\right) + B_1\left(U_{i+1,j+1}^{k+1} - U_{i,j}^{k+1}\right) + B_2\left(U_{i-1,j-1}^{k+1} - U_{i,j}^{k+1}\right) - B_3\left(U_{i+1,j-1}^{k+1} - U_{i,j}^{k+1}\right) - B_4\left(U_{i-1,j+1}^{k+1} - U_{i,j}^{k+1}\right)\Big) + U_{i,j}^{k+1} * I_{2x}^2 + V_{i,j}^{k+1} * I_{2y} * I_{2x}$$

Finalmente:

$$U_{i,j}^{k+1}\left(\frac{1}{\tau} + I_{2x}^2 + \mathrm{C}(A_1 + A_2 + C_1 + C_2 + B_1 + B_2 - B_3 - B_4)\right) + V_{i,j}^{k+1} * I_{2y} * I_{2x} - \mathrm{C}\Big(A_1\left(U_{i+1,j}^{k+1}\right) + A_2\left(U_{i-1,j}^{k+1}\right) + C_1\left(U_{i,j+1}^{k+1}\right) + C_2\left(U_{i,j-1}^{k+1}\right) + B_1\left(U_{i+1,j+1}^{k+1}\right) + B_2\left(U_{i-1,j-1}^{k+1}\right) - B_3\left(U_{i+1,j-1}^{k+1}\right) - B_4\left(U_{i-1,j+1}^{k+1}\right)\Big) = U_{i,j}^{k}\left(I_{2x}^2 + \frac{1}{\tau}\right) + I_{2x}\left(I_1 - I_2 + V_{i,j}^{k} * I_{2y}\right)$$

La estructura de la ecuación de los términos verticales, se parece a la primera. Hay que reemplazar la U por la V. Por tanto, repetimos los pasos anteriores:

$$\frac{V_{i,j}^{k+1} - V_{i,j}^{k}}{\tau} = \mathrm{C}(A_1 * V_1 + A_2 * V_2 + C_1 * V_3 + C_2 * V_4 + B_1 * V_5 + B_2 * V_6 - B_3 * V_7 - B_4 * V_8) + I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)\left(I_1\left(\bar{x}_{i,j}\right) - I_2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) + U_{i,j}^{k} * I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) + V_{i,j}^{k} * I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)\right) - V_{i,j}^{k+1} * I_{2y}^{2}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) - U_{i,j}^{k+1} * I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) * I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$

Renombramos otra vez:

$$I_1 = I_1\left(\bar{x}_{i,j}\right) \qquad I_2 = I_2\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$
$$I_{2x} = I_{2x}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right) \qquad I_{2y} = I_{2y}\left(\bar{x}_{i,j} + \bar{h}_{i,j}^{k}\right)$$

Queda

$$\frac{V_{i,j}^{k+1} - V_{i,j}^{k}}{\tau} = \overbrace{\mathrm{C}(\dots)} + \underbrace{I_{2y}\left(I_1 - I_2 + U_{i,j}^{k} * I_{2x} + V_{i,j}^{k} * I_{2y}\right)}$$
$$\overbrace{- V_{i,j}^{k+1} * I_{2y}^{2} - U_{i,j}^{k+1} * I_{2y} * I_{2x}}$$

Los miembros del lado derecho de la ecuación que tiene la llave por encima son términos de índice k+1. Los demás son términos k.

Ahora despejamos los términos k+1 a la izquierda:

$$\frac{V_{i,j}^{k+1}}{\tau} - \mathrm{C}(\dots) + V_{i,j}^{k+1} * I_{2y}^{2} + U_{i,j}^{k+1} * I_{2y} * I_{2x} = V_{i,j}^{k}\left(I_{2y}^{2} + \frac{1}{\tau}\right) + I_{2y}\left(I_1 - I_2 + U_{i,j}^{k} * I_{2x}\right)$$

Reescribimos de forma extendida el término izquierdo para sacar factor común (términos i,j):

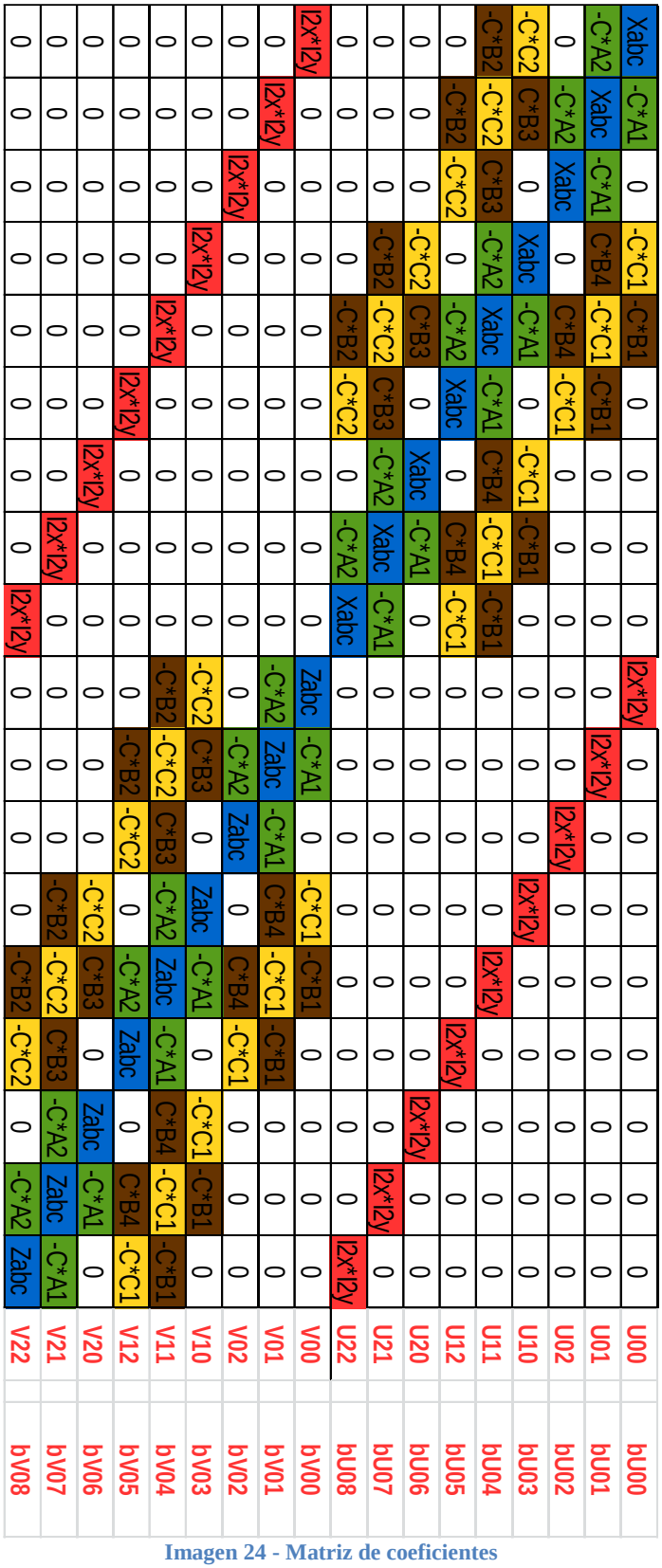
$$\frac{V_{i,j}^{k+1}}{\tau} - \mathrm{C}(\dots) + V_{i,j}^{k+1} * I_{2y}^2 + U_{i,j}^{k+1} * I_{2y} * I_{2x}$$

Finalmente

$$V_{i,j}^{k+1}\left(\frac{1}{\tau} + I_{2y}^2 + \mathrm{C}(A_1 + A_2 + C_1 + C_2 + B_1 + B_2 - B_3 - B_4)\right)$$
$$+U_{i,j}^{k+1} * I_{2y} * I_{2x} - \mathrm{C}\Big(A_1\big(V_{i+1,j}^{k+1}\big) + A_2\big(V_{i-1,j}^{k+1}\big) + C_1\big(V_{i,j+1}^{k+1}\big) +$$
$$C_2\big(V_{i,j-1}^{k+1}\big) + B_1\big(V_{i+1,j+1}^{k+1}\big) + B_2\big(V_{i-1,j-1}^{k+1}\big) - B_3\big(V_{i+1,j-1}^{k+1}\big) -$$
$$B_4\big(V_{i-1,j+1}^{k+1}\big)\Big) = V_{i,j}^{k}\left(I_{2y}^2 + \frac{1}{\tau}\right) + I_{2y}\big(I_1 - I_2 + U_{i,j}^{k} * I_{2x}\big)$$

Forma matricial

Continuando con el proceso de adaptación a C++, el siguiente paso consiste en colocar cada uno de los coeficientes en sus correspondientes posiciones dentro de la matriz densa que llamaremos A. Para esto, se muestra una matriz simplificada resultante de una imagen diminuta de dimensión 3x3:



Xabc	-C*A1	0	-C*C1	-C*B1	0	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	0	0	0	U00	bU00
-C*A2	Xabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	0	0	U01	bU01
-C*C2	C*B3	-C*A2	Xabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	0	U02	bU02
0	-C*C2	C*B3	-C*A2	Xabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	U10	bU03
-C*B2	-C*C2	C*B3	-C*A2	Xabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	U11	bU04
-C*B2	-C*C2	C*B3	-C*A2	Xabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	U12	bU05
-C*B2	-C*C2	C*B3	-C*A2	Xabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	U20	bU06
-C*B2	-C*C2	C*B3	-C*A2	Zabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	U21	bU07
-C*B2	-C*C2	C*B3	-C*A2	Zabc	-C*A1	C*B4	-C*C1	-C*B1	0	0	I2x*I2y	0	0	0	0	0	0	0	0	0	0	U22	bU08
I2x*I2y	0	0	0	0	0	0	0	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	0	0	0	V00	bV00
0	I2x*I2y	0	0	0	0	0	0	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	0	0	V01	bV01
0	0	I2x*I2y	0	0	0	0	0	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	0	V02	bV02
0	0	0	I2x*I2y	0	0	0	0	-C*B2	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	V10	bV03
0	0	0	0	I2x*I2y	0	0	0	-C*B2	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	V11	bV04
0	0	0	0	0	I2x*I2y	0	0	-C*B2	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	V12	bV05
0	0	0	0	0	0	I2x*I2y	0	-C*B2	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	V20	bV06
0	0	0	0	0	0	0	I2x*I2y	-C*B2	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	V21	bV07
0	0	0	0	0	0	0	0	-C*B2	-C*C2	-C*A2	Zabc	-C*A1	-C*C1	-C*B1	0	0	0	0	0	0	0	V22	bV08

Imagen 24 - Matriz de coeficientes

Con este paso, resalta a simple vista un problema incipiente: la complejidad espacial. Tan sólo por una imagen teórica y diminuta de 3 x 3 pixeles, resulta una

matriz cuadrada de dimensión 18 x 18 donde más del 60% de los elementos es cero, lo que significa que se está produciendo un desaprovechamiento muy importante del espacio.

La justificación de por qué una matriz tan pequeña desemboca en una de dimensiones mucho mayores se haya en el hecho de que para cada pixel es necesario calcular las componentes u y v. Por tanto, si la matriz es de 9 pixeles en total (3x3), deben calcularse 18 (3x3x2) elementos y dado que el sistema es cuadrado deben generarse 18 ecuaciones por lo que la matriz resultante será de 18 x 18.

Se hace necesario, por tanto, encontrar una forma de reducir esta complejidad dado que la tarea que se necesita resolver no resulta trivial.

Matriz escasa

El método elegido para subsanar el problema anteriormente mencionado ha sido disponer los elementos dentro de una matriz escasa (sparse). Una matriz escasa es aquella en la que la mayoría de sus elementos son cero.

Existen muchos métodos alternativos para representar una matriz escasa con sus pros y sus contras. Pero, por cuanto, en este proyecto el factor importante es la reducción espacial se ha elegido el formato CSR (compressed row storage; almacenamiento de filas comprimido) el cual es de tipo general lo que quiere decir que no está pensado ni enfocado para ningún tipo de matriz en especial a la vez que no almacena elementos innecesarios.

CRS coloca los subsiguientes elementos no nulos (distintos de cero) de las filas de la matriz en ubicaciones de memoria contiguas. Para ello se vale de 3 vectores que harán las veces de matriz, índice de columnas y contador de elementos por filas respectivamente. El primero almacenará los elementos no nulos de la matriz original mientras que el segundo contendrá los índices de las columnas de cada elemento del primer vector. Finalmente el tercer vector (cuya dimensión es igual al número de filas más uno) contendrá las posiciones donde comienza cada fila siendo indicadas por la diferencia entre el siguiente elemento y el actual. A continuación se muestra un ejemplo.

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

$A = (10 \quad -2 \quad 3 \quad 9 \quad 3 \quad 7 \quad 8 \quad 7 \quad 3 \quad 8 \quad 7 \quad 5 \quad 8 \quad 9 \quad 9 \quad 13 \quad 4 \quad 2 \quad -1)$
$Col. = (1 \quad 5 \quad 1 \quad 2 \quad 6 \quad 2 \quad 3 \quad 4 \quad 1 \quad 3 \quad 4 \quad 5 \quad 2 \quad 4 \quad 5 \quad 6 \quad 2 \quad 5 \quad 6)$
$Filas = (1 \quad 3 \quad 6 \quad 9 \quad 13 \quad 17 \quad 20)$

De esta forma, el ahorro de espacio es muy significativo dado que la complejidad se la complejidad se reduce de $\Theta(n^2)$ a $\Theta(n)$ porque solamente se necesitan almacenar 2 * enn + n + 1, siendo enn el número de elementos no nulos y n el número de filas de la matriz:

Nº de ecuaciones	2 * n * m
Dimensión matriz densa	(2 * n * m)^2
Dimensión matriz escasa	2 * (20 + (n - 2) * (m - 2) * 10 + 2 * (n - 2) * 7 + 2 * (m - 2) * 7)

Tabla 24 - Relación Ecuaciones vs Dimensión

Por consiguiente, el sistema de ecauciones correspondiente a una matriz de 3 x 3 como la descrita anteriormente, sería:

		0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8
		U00	U01	U02	U10	U11	U12	U20	U21	U22	V00	V01	V02	V10	V11	V12	V20	V21	V22
0	U00	0	1		3	4					0								
1	U01	0	1	2	3	4	5					1							
2	U02		1	2		4	5						2						
3	U10	0	1		3	4		6	7					3					
4	U11	0	1	2	3	4	5	6	7	8					4				
5	U12		1	2		4	5		7	8						5			
6	U20				3	4		6	7								6		
7	U21				3	4	5	6	7	8								7	
8	U22					4	5		7	8									8
0	U00	0									0	1		3	4				
1	U01		1								0	1	2	3	4	5			
2	U02			2								1	2		4	5			
3	U10				3						0	1		3	4		6	7	
4	U11					4					0	1	2	3	4	5	6	7	8
5	U12						5					1	2		4	5		7	8
6	U20							6						3	4		6	7	
7	U21								7					3	4	5	6	7	8
8	U22									8					4	5		7	8

Imagen 25 - Matriz de ejemplo (3x3)

Y se representaría de la siguiente manera en el formato CSR:

A	0	1	3	4	0	0	1	2	3	4	5	1	***					
col_ind	0	1	3	4	9	0	1	2	3	4	5	10	***					
row_ptr	0	5	12	17	24	34	41	46	53	58	63	70	***					
	U00	U01	U02	U10	U11	U12	U20	U21	U22	V00	V01	V02	V10	V11	V12	V20	V21	V22

Imagen 26 - Matriz CSR

Matriz escasa en CUDA

La representación de la matriz, en la versión secuencial, se ha hecho en forma de vector igual que en las funciones paralelizadas que se ejecutan en los núcleos de la tarjeta gráfica con el fin de simplificar el tratamiento de la información y la programación de la aplicación. De manera que para direccionar cualquier elemento de la matriz, siendo ésta representada en forma de array lineal y por consiguiente utilizando un solo bucle en lugar de dos, para calcular su posición basta con traducir el índice actual del bucle a la correspondiente fila siendo esta el cociente entre dicho índice y el valor de la ancho de la matriz a la vez que el resto de esa división será el desplazamiento horizontal, es decir, la columna en la que se encuentra el elemento.

Esto se ha hecho así con el fin de minimizar los accesos a memoria, sobre todo en CUDA porque dada su arquitectura, la memoria ha de estar alineada y esto implica que no hay que realizar ninguna acción cuando se trata de memoria unidimensional pero todo lo contrario en el resto de los casos.

El problema consiste en que cuando se trata con memoria en más de una dimensión, como es el caso de este proyecto, el número de operaciones de acceso a memoria necesarios depende directamente de la cantidad de palabras que necesita cada fila (de bytes) y el número de bytes de cada palabra de memoria depende de la implementación. Para reducir al mínimo dicho número de accesos cuando se lee una fila, se ha de garantizar que el proceso de lectura comienza al principio de cada palabra, por lo tanto, se debe rellenar la memoria de cada fila hasta el comienzo de una nueva.

Además, puede ocurrir que dos hilos accedan a la misma posición de memoria (compartida) de forma simultánea y teniendo en cuenta que, por lo

general, el objetivo es tratar cada una en paralelo, es menester rellenar y desplazar cada cual al inicio de un nuevo banco.

Por otra parte, dicho desplazamiento (en bytes) ya no podrá ser deducido a través del tamaño de la matriz; es por eso que resulta imperativo mantener el campo devuelto por cudaMallocPitch (indica el número de bytes que se ha desplazado al memoria) para acceder a cualquier posición.

Como se puede apreciar manejar memoria en más de una dimensión al trabajar con CUDA significa añadir una capa de complicación innecesaria a este proyecto. Es por este motivo que se ha simplificado la representación de las matrices como si fuesen vectores.

En la siguiente imagen se puede apreciar la implementación de los coeficientes obtenidos durante la formulación explícita del sistema (página 90) de ecuaciones propuesto en el documento:

```
// constant D coeff computation (computed from I1)
float A0 = Da[INDEX(i, j, N, M)]; // Aij
float B0 = Db[INDEX(i, j, N, M)]; // Bij
float C0 = Dc[INDEX(i, j, N, M)]; // Cij

// neighbors coefficients (equation in pdf)
float A1 = (Da[INDEX(i + 1, j, N, M)] + A0) * rxx; // E
float A2 = (Da[INDEX(i - 1, j, N, M)] + A0) * rxx; // W

float B1 = (Db[INDEX(i + 1, j + 1, N, M)] + B0) * rxy; // SE
float B2 = (Db[INDEX(i - 1, j - 1, N, M)] + B0) * rxy; // NW
float B3 = (Db[INDEX(i + 1, j - 1, N, M)] + B0) * rxy; // NE
float B4 = (Db[INDEX(i - 1, j + 1, N, M)] + B0) * rxy; // SW

float C1 = (Dc[INDEX(i, j + 1, N, M)] + C0) * ryy; // S
float C2 = (Dc[INDEX(i, j - 1, N, M)] + C0) * ryy; // N
// end of constant D coeff computation
```

Imagen 27 - Cálculo de los pesos de cada pixel

Los factores rxx y ryy valen 0.5 mientras que rxy vale la mitad, es decir, 0.25. Finalmente, si nos encontramos en alguna de las cuatro esquinas, el valor de los pesos asociados a los vecinos superiores será cero por cuanto no existen.

Además, estos parámetros dependen directamente de los valores resultantes al momento de calcular el difusor de tensión descrito por la siguiente ecuación:

$$D(\nabla I_1) = \frac{1}{\|\nabla x\|^2 + 2\zeta^2}\left(\begin{bmatrix} \frac{dI_1^2}{dy} & \frac{-dI_1}{dy}\frac{dI_1}{dx} \\ \frac{-dI_1}{dy}\frac{dI_1}{dx} & \frac{dI_1^2}{dx} \end{bmatrix} + \zeta^2 Id\right)$$

donde Da se corresponde con los términos asociados a la derivadar vertical, Db con el producto, y Dc con la horizontal.

El siguiente paso trata de la inserción de valores en el término independiente, b, y en la matriz de valores A (val en el fragmento de código):

```
float u = x[y], v = x[N * M + y];

float xDerivative = interpolate ((float *) XDerivative, i, j, I1.getWidth(), I1.getHeight(), u, v);
float yDerivative = interpolate ((float *) YDerivative, i, j, I1.getWidth(), I1.getHeight(), u, v);

float XABC = tau + cte * (A1 + A2 + C1 + C2 + B1 + B2 - B3 - B4) + xDerivative * xDerivative;

float interpolation = interpolate ((float *) GI2, i, j, I1.getWidth(), I1.getHeight(), u, v);

b[y] = u * (tau + xDerivative * xDerivative)
     + xDerivative * (GI1[INDEX(i, j, N, M)] - interpolation + v * yDerivative);

if (i - 1 >= 0 && j - 1 >= 0) val[k++] = - cte * B2;
if (i - 1 >= 0)               val[k++] = - cte * A2;
if (i - 1 >= 0 && j + 1 < M)  val[k++] =   cte * B4;
if (j - 1 >= 0)               val[k++] = - cte * C2;
                              val[k++] = 0.;
                              D[y]     = 1./XABC;
if (j + 1 < M)                val[k++] = - cte * C1;
if (i + 1 < N && j - 1 >= 0)  val[k++] =   cte * B3;
if (i + 1 < N)                val[k++] = - cte * A1;
if (i + 1 < N && j + 1 < M)   val[k++] = - cte * B1;
                              val[k++] = xDerivative * yDerivative;
```

Imagen 28 - Cálculo de los coeficientes de A

En la imágen precedente es importante prestar atención a tres aspectos importantes:

a. Tanto el valor de las derivadas como el pixel (i, j) de la segunda imagen se interpolan mediante un proceso bicúbico, dado que así se indica en la ecuación original.

b. Se asigna cero a la diagonal de la fila actual de la matriz de tal forma que no haya que hacer ningún tipo de preprocesamiento a la hora de resolver el sistema con el fin de obtener un menor tiempo de ejecución.

c. Jacobi require dividir el producto escalar por la diagonal, y dado que antes esta ha sido puesta a cero, se ha empleado un vector adicional que contiene la inversa de dicho valor así la función que haya la solución del sistema mediante el método de Jacobi sólo tiene que multiplicar este número por el producto de la fila.

El último paso, y más importante, consiste en resolver el sistema. En la versión secuencial la CPU se encarga de procesar todas las filas de la matriz, lo que puede verse en el bucle exterior, mientras que en el bucle interior se calcula el producto escalar de cada fila incluyendo la diagonal porque al haber sido inicializado su valor a cero, no afecta al resultado. Finalmente se multiplica la diferencia entre el término independiente y el producto por la inversa de la diagonal.

```
/*
 * @brief: jacobi carries out one step of the Jacobi iteration. --
 *
 *   Inputs:
 *    double a[N], the sparse (CSR) matrix.
 *    double b[N], the right hand side.
 *    double x[N], the current solution estimate.
 *    double xNew[N], the current solution estimate.
 *    int    row_ptr, the list of value indexes where each row starts.
 *    int    col_ind, the column indices corresponding to the values.
 *    int    N, the order of the matrix.
 */
template <typename T>
void jacobiIteration(T * A, T * x, T * xNew, T * b, T * d,
                     const int * row_ptr, const int * col_ind,
                     const int N) {
    float error = 0;
    for ( int i = 0; i < N; i++ )
    {
        T dot = 0.;
        for(int j = row_ptr[i]; j < row_ptr[i + 1]; j++)
            dot += A[j] * x[col_ind[j]]; // A == L + U

        xNew[i] = d[i] * (b[i] - dot);
    }
}
```

Imagen 29 - Iteración secuencial de Jacobi

El procedimiento es exactamente el mismo en la versión paralela con la salvedad de que cada hilo de ejecución se ocupa de una fila de tal manera que cada una se procesa de forma simultánea.

```
/**
 * @brief: runs one iteration of the jacobi method
 */
__global__ void jacobiIterationCUDA(float * A, float * x, float * xNew, float * b, float * d,
                                    const int * row_ptr, const int * col_ind, const int N)
{
    register int i = threadIdx.x + blockIdx.x * blockDim.x;

    if (i < N)
    {
        register int jbegin = row_ptr[i],
                     jend   = row_ptr[i + 1];

        register float dot = 0.;

        for (register int j = jbegin; j < jend; j++)
            dot += A[j] * x[col_ind[j]];

        xNew[i] = d[i] * (b[i] - dot);
    }
}
```

Imagen 30 - Iteración paralela de Jacobi

De esta forma, se ha procedido a realizar modificaciones en cada una de los problemas paralelizables en el código, excepto cuando el grado de paralelismo puede alcanzar el nivel de pixel, es decir, cuando cada elemento sólo depende de si mismo, como por ejemplo el pasar a escala de grises:

```
/**
 * Image to grayscale: 0.3 * r + 0.59 * g + 0.11 * b
 * @params:
 *  unsigned char *: r, g, b --> bmp channels
 **/
__global__ void grayScaleCuda(unsigned char * r,
                              unsigned char * g,
                              unsigned char * b,
                              float * outputChannel,
                              int dim)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < dim)
    {
        outputChannel[ i ] = 0.3 * r[ i ] + 0.59 * g[ i ] + 0.11 * b[ i ];
    }
}
```

Imagen 31 - Conversión a escala de grises

donde se crean tantos hilos como pixeles para aumentar la productividad ya que el procesamiento de cada uno es independiente del resto.

Resultados y conclusiones

Tras el trabajo realizado a lo largo de todo el proyecto fin de carrera, en el que se han llevado a cabo fases de estudio, análisis, diseño y desarrollo, se han obtenido una serie de resultados de interés y se ha podido llegar a diversas conclusiones importantes, presentadas en los siguientes apartados.

Resultados

Los resultados presentados a continuación son el fruto de más de 800 horas de trabajo a lo largo de un curso académico. Con más de 30 ficheros desarrollados y organizados en diferentes por módulos, compuestos por más de 4500 líneas de código implementadas en C y C++.

Entorno gráfico

Uno de los aspectos fundamentales que se pretendía conseguir con el desarrollo de la aplicación de ventanas era facilitar la comunicación hombre máquina, es decir, proveer de una herramienta mucho más sencilla e intuitiva para poder analizar los resultados a la vez que aportase mayor flexibilidad y permitiese trabajar con distintos tipos de ficheros de una forma más cómoda.

La posibilidad de cargar tanto imágenes como secuencias de video, la capacidad almacenar los resultados tanto numéricos como gráficos, las herramientas de análisis de las características de los frames como de las propiedades del hardware disponible, la visualización numérica del valor del deplazamiento particular de cada pixel así como el poder elegir el formato de representación del vector de flujo para poder entender el movimiento a la par que una interfaz simple e intuitiva y sobre todo el disponer del software para acelerar los cálculos componen un sistema de investigación que cubre un amplio espectro de necesidades en los que prima la precisión y la calidad.

Datos empíricos

En este apartado se expondrán los resultados experimentales obtenidos en los diversos experimentos a la vez que se realizará una comparación del

mismo para establecer el porcentaje de mejora que ofrece la versión paralela con su respectiva homóloga secuencial.

Para poder cuantificar la mejora se ha medido el tiempo de ejecución de las secciones más relevantes del algoritmo tales como los encargados de procesar la derivada horizontal y vertical, el gradiente, el método de Jacobi, la convolución gausiana, la matriz D y la construcción del sistema de ecuaciones atendiendo a distintos tamaños de imagen junto con la variación del radio empleado para calcular el filtro gausiano.

Con el objetivo de realizar una comparación clara y equitativa, se ha medido el rendimiento de los métodos cambiando el valor de los parámetros uno por uno obteniendo el promedio de cada uno de las medidas anteriormente mencionadas durante 50 pruebas aleatorias diferentes.

Se usaron pares de frames de los siguientes tamaños: 256, 512 y 1024.

La primera parte ha consistido en comprobar la calidad de los resultados no varía independientemente de su origen para lo que se calculó la diferencia de cada componente y se comprobó que la diferencia es menor que 1E-3.

En cuánto al rendimiento, los resultados presentan el tiempo en milisegundos requerido por cada programa para realizar solamente la creación del sistema de ecuaciones y la ejecución del algoritmo de flujo óptico, es decir, excluyendo el tiempo empleado para la lectura de imágenes, la preparación de la memoria de la tarjeta gráfica y visualización de resultados.

Como referencia base se utilizarán los valores procedentes de ejecutar el algoritmo con un radio de valor 4 para la convolución gausiana. En las siguientes dos tablas se aprecia la clasificación de los resultados según los tiempos de ejecución acorde al modo (secuencial o paralelo) y a la resolución de la imagen.

CPU

Tamaño	Total	Dx	Dy	Gradiente	Jacobi	Gauss	Matriz D	Sist. Ec.
256	1832,6	5,18736	5,17954	7,60285	67,897	500,405	6,72666	729,7
512	7453,8	21,6819	21,6756	30,5885	260,17	2028,96	30,1762	3017,34
1024	30955,3	91,5432	91,3475	129,331	925,129	8298,73	163,89	12906,2

Table 1 - Resultados CPU - Gauss(4)

GPU

Tamaño	Total	Dx	Dy	Gradiente	Jacobi	Gauss	Matriz D	Sist. Ec.
256	966,415	1,82072	1,82041	1,54608	22,6925	34,2268	2,22405	736,732
512	4094,62	6,46032	6,45903	4,81222	90,5873	150,235	7,07027	3195,37
1024	17097,2	25,1082	25,1016	20,3673	382,584	855,205	27,4247	12912,6

Table 2 - Resultados GPU - Gauss(4)

Y, junto a las tablas, un gráfico de ganancias donde se representan la información mostrada arriba de la siguiente manera:

- Las 3 primeras columnas consisten en los tiempos de ejecución totales extraídos de la CPU mientras que las de la derecha provienen de la GPU.

- Dentro de cada uno de estos dos grupos cada una de las tres columnas se corresponde con cada uno de los tamaños de imagen con los que se realizaron las pruebas.

- A su vez, cada columna, presenta subregiones de distintos colores cuyo significado se explicará debajo.

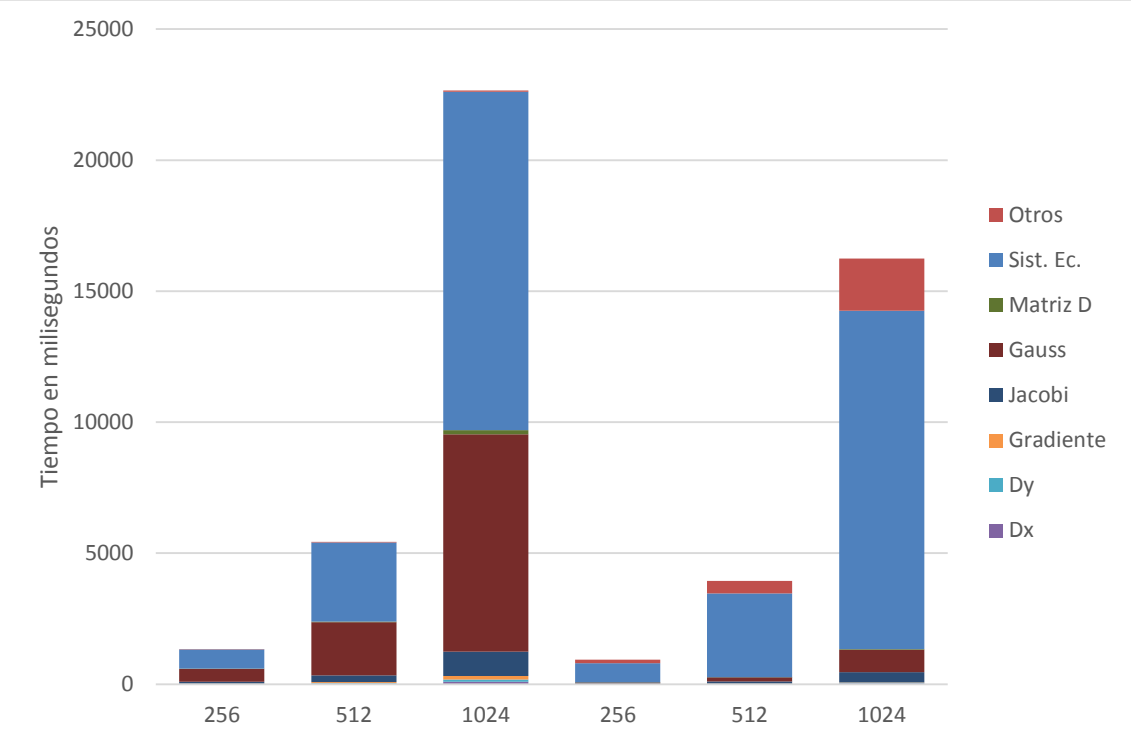


Gráfico 1 - Reparto de tiempos según modo de ejecución

Lo que más notablemente se distingue en las columnas es la sección azul. Esta corresponde al tiempo total necesario para generar el sistema de ecuaciones durante todas las iteraciones llevadas a cabo durante el proceso. El hecho de que sea tan considerable se debe a que la construcción de dicho sistema se realiza de forma secuencial al estar este contenido en una matriz escasa de tipo CSR donde el relleno de los coeficientes es muy difícil de paralelizar, sino imposible, lo cual puede verse claramente en la Imagen 28 - Cálculo de los coeficientes de A. El porqué de esto radica en la complejidad asociada a desacoplar la colocación de los índices y coeficientes dado que la matriz final se compone en realidad de tres vectores: índices, columnas y coeficientes. Por esto, estos tiempos aunque computan para la cifra total no han de tenerse en cuenta para la obtención de las ganancias, si las hubiere, en la versión paralela. Además, otra subtarea que consume demasiado tiempo dentro de esta sección es la que se ocupa de interpolar la nueva posición de cada pixel dentro de la segunda imagen usando el nuevo valor de flujo extraido de resolver el sistema en la última iteración.

La franja morada muestra la cantidad de tiempo que necesita el programa para realizar el difuminado según el tamaño de imagen. Y es aquí donde se aprecia que el factor de aceleración permanece constante con los dos primeros tamaños de image pero que decrece aproximadamente un 40% cuando la imagen supera el millón de píxeles. De hecho, para este tamaño de imagen para el tramo de sigma entre 1 y 4 la ganacia ronda en torno a 9,8 mientras que de 8 a 32 dicha ganancia sube a 12,1. Esta diferencia puede deberse probablemente conflictos en el patrón de acceso a la memoria global donde reside la imagen.

Cabe destacar que si bien los tiempos de ejecución de las dos tablas que preceden al gráfico son correctos, debe tenerse en cuenta que el tiempo total incluye también la duración del desenfoque gausiano aplicado a la segunda imágen, es decir, que las estadísticas sólo contemplan el proceso de Gauss una vez aunque se aplica dos veces (imagen 1 e imagen 2). Por tanto, en el gráfico, para representar la información de forma coeherente no se contempla la segunda ejecución y se ha restado de la sección “Otros” de las columnas.

Seguidamente puede verse el impacto directo de la convolución gaussiana en el tiempo total y cómo, cuando empleamos la GPU, el factor de aceleración se incrementa a medida que el radio de la vecindad de píxeles también lo hace.

Como se aprecia en el gráfico que se encuentra debajo de la tabla, el tiempo necesario para realizar la convolución aumenta de forma sostenida y proporcionalmente al valor de la sigma elegida (sección amarilla). Esto asimismo contrasta con su versión paralela que presenta tiempos de ejecución ostensiblemente menores siendo de media 13,5 veces más rápido.

		CPU		GPU	
Tamaño	Sigma	Total	Gauss	Total	Gauss
256	1	179,921	16,8388	150,545	1,07517
	2	713,688	121,024	522,351	8,14685
	4	1832,6	500,405	966,415	34,2268
	8	5252,4	2040,62	1530,34	133,561
	16	17527,4	7991,82	2687,21	515,465
	32	71152,4	34441,6	6202,33	2048,14
512	1	674,592	57,8935	605,57	4,3345
	2	2849,31	465,61	2132,3	34,1596
	4	7453,8	2028,96	4094,62	150,235
	8	23819,6	9196,05	6337,76	605,338
	16	82128,4	37266,8	11856	2473,89
	32	295864	142276	27746,4	9414,9
1024	1	2775,15	235,178	2608,31	23,6444
	2	11871,2	1905,05	9207,35	195,432
	4	30955,3	8298,73	17097,2	855,205
	8	102007	38928,1	29989,5	3226,66
	16	329838	148580	54522,8	12456,4
	32	1273940	613668	142662	49712

Table 3 - Tiempo total vs Tiempo de Gauss

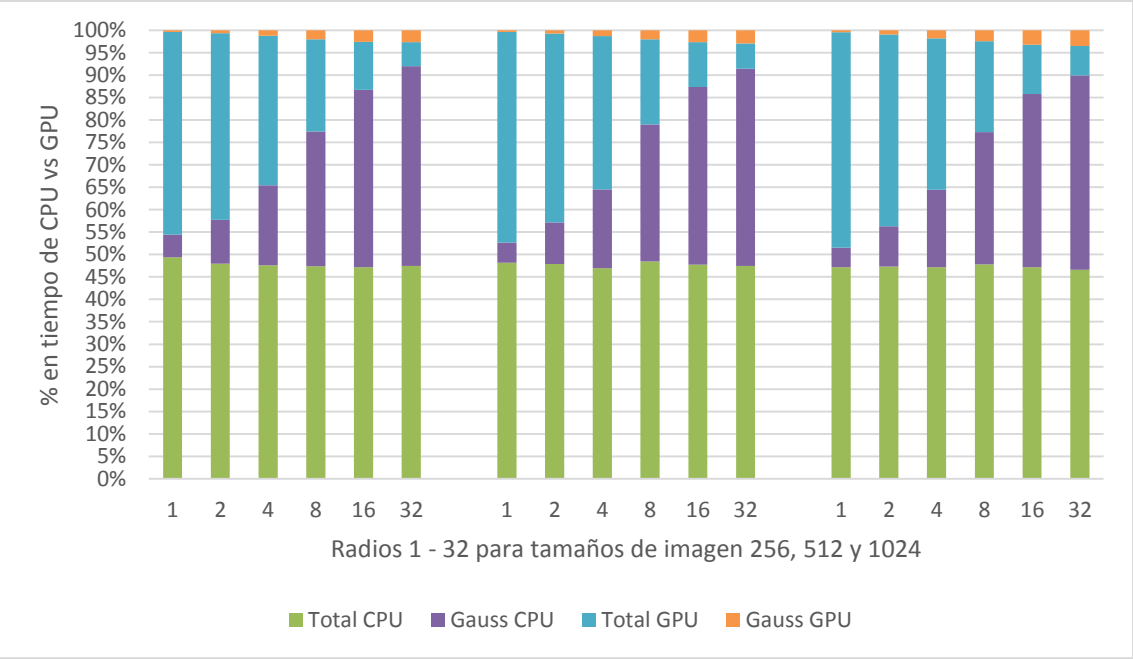
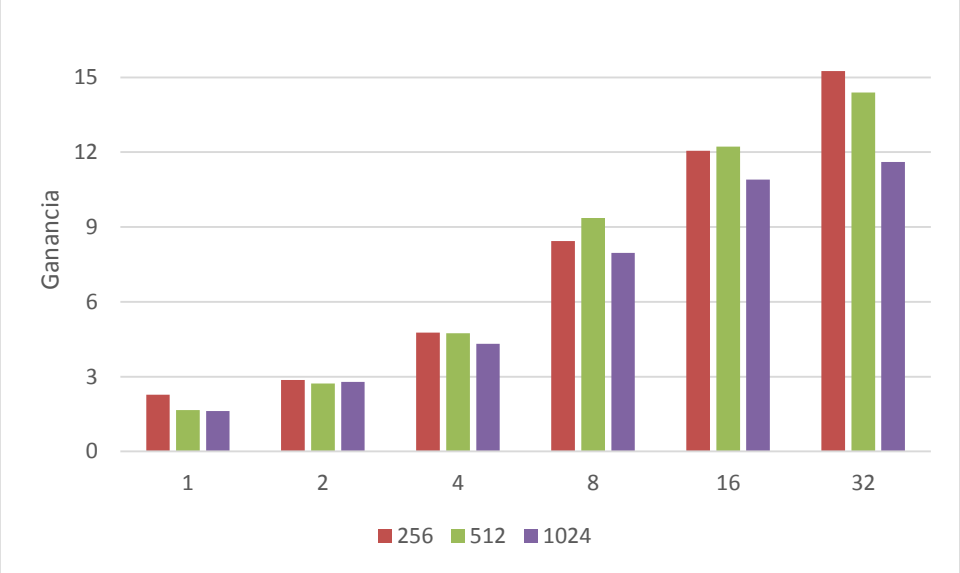


Gráfico 2 - Impacto de Gauss en el tiempo total

Finalmente, se muestran las ganancias totales de la versión paralela con respecto a la secuencial excluyendo, como se explicó en "Gráfico 1 - Reparto de tiempos según modo de ejecución" los tiempos comunes que son los necesarios para construir el sistema de ecuaciones para cada pixel.

Tamaño	1	2	4	8	16	32
256	2,277	2,869	4,771	8,437	12,060	15,256
512	1,656	2,719	4,736	9,362	12,229	14,398
1024	1,620	2,794	4,312	7,960	10,904	11,603

Table 4 - Ganacias totales



Lamentablemente, aunque las ganancias totales crecen junto con el tamaño de imagen y el valor de sigma, a simple vista, el usuario, no percibe una mejora considerable en el resultado final porque, aunque por separado los componentes si se benefician de la GPU, el programa no puede obviar el hecho de que debe reconstruir el sistema de ecuaciones cada vez que se modifica el radio de la vecindad de píxeles a la par que para esto también debe interpolar la nueva posición de cada pixel, todo lo cual se realiza de forma secuencial ya que el formato de matriz escasa elegido presenta esta limitación.

No obstante, el uso de la GPU es adecuada para el problema, ya que en cada una de las pruebas realizadas, la implementación basada en paralelización fue más rápida.

Análisis del rendimiento

La causa de que la ganancia en velocidad de la versión paralela del algoritmo no sea mayor que la presentada se debe a que hay conflictos en los bancos de memoria dado que la obtención de la información no se realiza de forma simultánea, es decir, no es coalescente. La razón de esto es el patrón de acceso el cual es directamente dependiente de la estructura de la matriz escasa (CSR).

Antes de continuar, combiene definir la limitación por latencia y por cómputo.

Un núcleo limitado por la latencia es aquel cuyo motivo predominante en la pérdida de rendimiento son los accesos a memoria, en concreto las instrucciones de tipo fetch. Así que, aunque el bus no se encuentre saturado, gran parte del retardo se debe a la espera de los datos.

Por otro lado, los limitados por el grado o la cantidad de cómputo presentan como característica principal que su tiempo de ejecución está acaparado fundamentalmente por operaciones de cálculo.

Como se demostrará más abajo, el problema inherente al rendimiento depende directamente del tiempo de ejecución del núcleo que realiza las iteraciones de jacobi donde se puede decir sin miedo a equivocarse que se trata de un problema limitado por la latencia dado que consiste en, a excepción de una división, calcular el producto escalar entre una matriz

(escasa) y un vector (sólo multiplicaciones y sumas), por lo que maximizar el uso de memoria es clave para optimizar el rendimiento de la GPU.

Las GPUs NVIDIA están diseñadas para ser muy productivas gracias a sus muchos procesadores que ofrecen un rendimiento computacional muy alto. Aprovechar todo este potencial requiere exponer grandes cantidades de paralelismo de grano fino (dividir un problema muy grande en subtareas pequeñas) y disponer los cálculos de tal manera que las rutas de ejecución sean lo suficientemente regulares (la divergencia de los hilos sea muy pequeña) y que los patrones de acceso a memoria sean eficientes.

Un warp ejecuta una sola instrucción a la vez en todos sus hilos. Los hilos de un warp son libres de seguir su propio camino de ejecución. Sin embargo, es sustancialmente más eficiente para los hilos, que todos sigan el mismo camino durante la mayor parte del cálculo.

El acceso a ubicaciones dispersas resulta en divergencia de memoria y requiere que el procesador realice una transacción de memoria por hilo. Por otro lado, si las ubicaciones a las que se accede están suficientemente cerca entre sí, las operaciones por hilo pueden unirse para una mayor eficiencia de memoria

El acceso no contiguo a la memoria es perjudicial para la eficiencia de ancho de banda de memoria y por lo tanto también para el rendimiento de los núcleos de memoria.

Formato CSR (Compressed Sparse Row)

Puesto que el producto escalar entre una fila de la matriz y el vector puede calcularse independientemente de todas las demás filas, la operación de jacobi es fácilmente paralela. Mientras que el núcleo implementado en este proyecto muestra paralelismo de grano fino, su rendimiento sufre de varios inconvenientes. El más significativo es la manera en que los hilos acceden a los índices CSR y a los arrays de datos. A pesar de que los índices de columna y los valores distintos de cero para una fila dada se almacenan de forma contigua, no se acceden simultáneamente. En su lugar, cada hilo lee los elementos de su fila secuencialmente, produciendo el patrón mostrado en la siguiente figura (suponiendo una matriz de ejemplo como la siguiente):

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

índices	0	1	1	2	0	2	3	1	3
Info	1	7	2	8	5	3	9	6	4
iter 1	0		1		2			3	
iter 2		0		1		2			3
iter 3							2		

El problema

En otras palabras, como puede verse en la imagen a continuación, en el formato CSR las filas de la matriz se colocan de forma secuencial en el array lo cual es un inconveniente dado que cada hilo se encarga de una fila:

Imagen 32 - Matriz esacasa CSR

Los problemas evidentes son:

- mal o poco acceso coalescente
- accesos a memoria desalienados

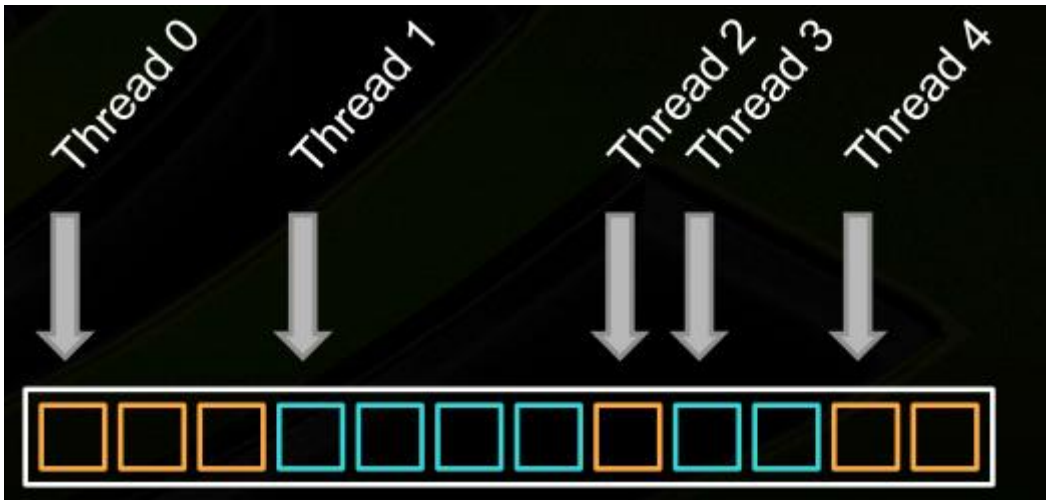


Imagen 33 - Acceso no coalescente

Como se mencionó anteriormente, hay que tener en cuenta que todos los hilos pertenecientes a un warp ejecutan la misma instrucción. Esto implica, por ejemplo, que cuando dicha instrucción es de carga, el hardware detecta si los accesos pertenecen a direcciones consecutivas. Así que el proceso se desarrollará mejor cuando el hardware combine todos estos accesos en un solo, conectando así con direcciones correlativas, es decir, si el hilo 0 accede a la ubicación n, el hilo 1 accede a la ubicación n + 1, así hasta el hilo 31 que accede a la posición n + 31.

Acceso coalescente

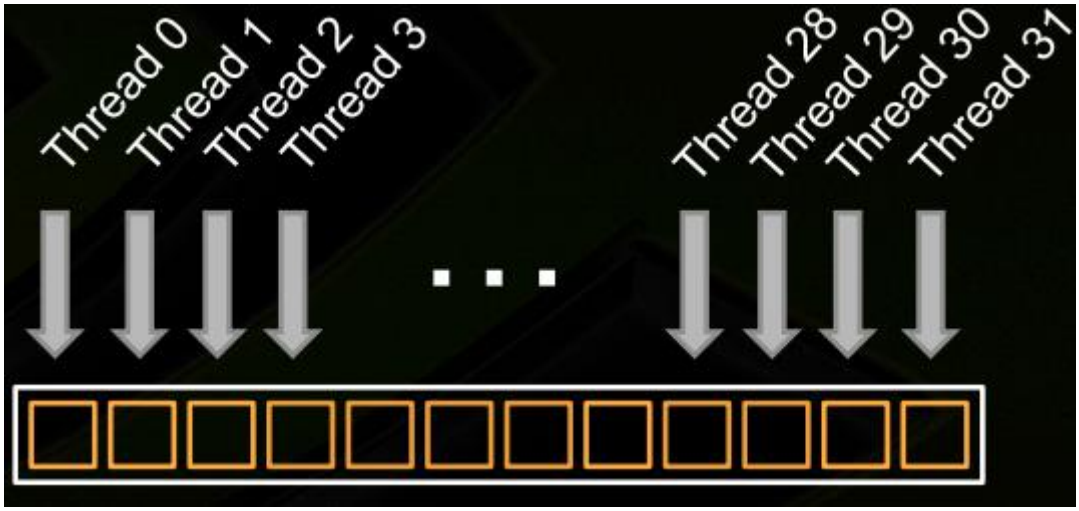


Imagen 34 - Acceso coalescente

Al comparar el patrón de acceso de la matriz con el de la imagen precedente se puede apreciar que se producen conflictos en los bancos de memoria por cuanto, durante la ejecución, al menos dos hilos acceden a direcciones presentes en el mismo banco y sabiendo que cada banco sólo puede servir una dirección por ciclo, el resultado es que las lecturas son serializadas, o dicho de otra forma, se realizan de forma secuencial.

Solución propuesta

Conocemos que el maximo numero entradas distintas de cero en cada fila de la matriz escasa será 10. De hecho, se conoce de antemano cuántos elementos por fila habrá. Si se toma como ejemplo una matriz de 6x6 como la siguiente, la matriz escasa presentará 5 elementos en las filas correspondientes a los pixeles azules, 7 en los anaranjados y 10 en el resto. La ecuación de cada pixel contiene su valor más el de sus vecinos y el de la componente vertical u horizontal según el caso.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Por tanto, empleando un poco más de espacio, se puede añadir un pequeño padding en cada fila, cambiando el formato de la matriz, pasando del formato CSR al ELL el cual consiste en el uso de dos matrices con el mismo número de filas que la matriz densa original pero solamente con tantas columnas como el número máximo de elementos no nulos por fila, es decir, 10 en nuestro caso.

De esta forma, la nueva matriz escasa, aunque requeriría un poco más de memoria, presentaría una estructura fija con la que poder realizar un acceso completamente coalescente de tal forma que los conflictos en los módulos de memoria se reduzcan.

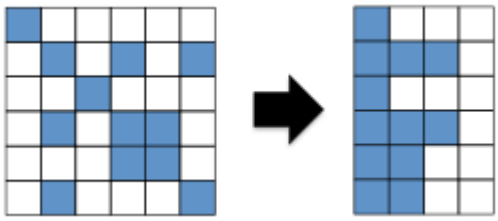


Imagen 35 - Ejemplo de matriz ELL

A su vez, también se efectuaría un relleno en la altura de la matriz de tal forma que sea múltiplo de 8

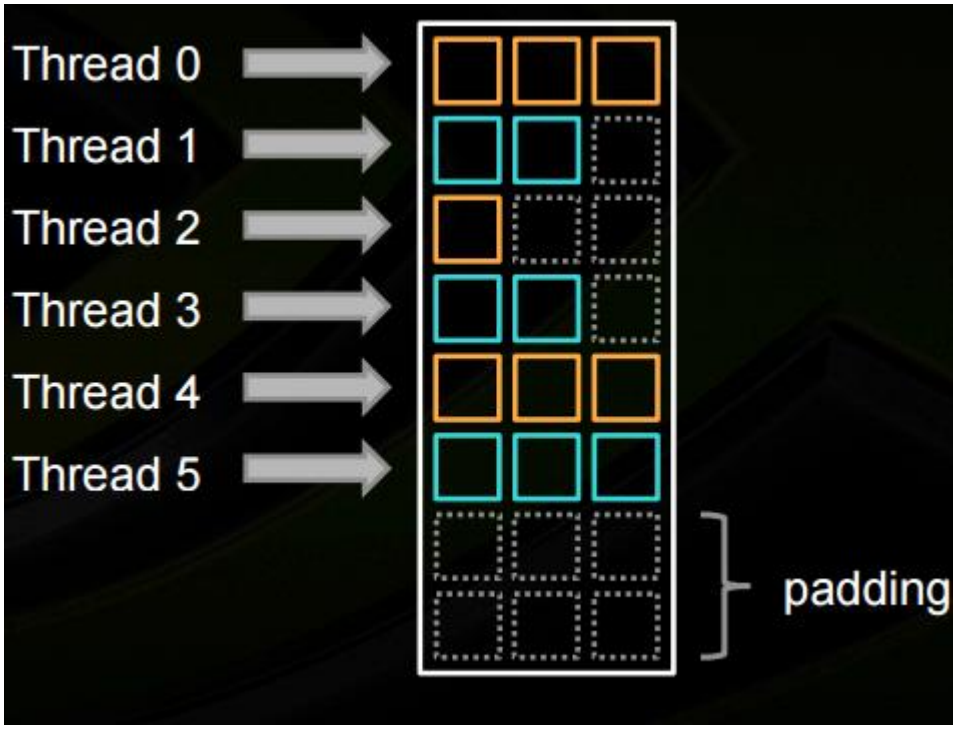


Imagen 36 - ELL Padding

De esta manera, se conseguiría que la lectura de los datos fuesen contiguas en cada iteración:

índices	0	1	0	6	1	2	2	4	*	*	3	*
Info	1	2	5	6	7	8	3	4	*	*	9	*
iter 1	0	1	2	3								
iter 2					0	1	2	3				
iter 3									0	1	2	3

Conclusiones

Un aspecto fundamental a la hora de dar por finalizado un proyecto de fin de carrera es comprobar que los objetivos marcados en el comienzo del mismo se han satisfecho con el trabajo desarrollado.

En este caso se han podido cumplir todos los aspectos propuestos al inicio.

En este trabajo, se ha conseguido realizar una implementación eficiente del algoritmo de flujo óptico elegido. Más precisamente, se ha desarrollado un software basado en GPU que aplica el método de Nagel-Enkelmann con mejoras al par de frames y/o cada par de frames de una secuencia de video, previamente seleccionados. Además, el método está optimizado para la arquitectura de Fermi, especialmente en el contexto del acceso a la memoria.

Las pruebas realizadas muestran que la herramienta propuesta es útil tanto para realizar pruebas en fase de experimentación así como de investigación.

Gracias a la realización de este proyecto se han podido estudiar nuevas tecnologías y técnicas de programación debido a la necesidad de gestionar estructuras de datos e hilos de forma que se pueda aprovechar todo el potencial de la paralelización. No sólo ha sido necesario conocer C/C++, sino también las metodologías y estrategias de la plataforma CUDA. Debido a esta circunstancia, el trabajo realizado para el desarrollo se ha enriquecido enormemente.

Además, al haber conseguido implementar una herramienta gráfica que alberga la capacidad de realizar cómputos en paralelo junto con la habilidad de procesar imágenes, se ha adquirido un conjunto de habilidades nuevas que sin duda podrán ser utilizadas en un futuro tanto para poder mejorar las metodologías de trabajo empleadas de manera personal como para comenzar nuevos proyectos de diferentes características.

Por otro lado, a pesar de no conocer todas las singularidades del proyecto o los posibles problemas que pudieran aparecer durante su desarrollo, ha sido fundamental la capacidad de saber retroceder cuando ha sido necesario y volver a comenzar por otra vía. De esta manera, se ha podido encontrar solución a los impedimentos que han surgido durante el proceso.

Sin lugar a dudas, la realización de un proyecto de fin de carrera de estas características requiere una gran dedicación y esfuerzo, pero se ven recompensados por la satisfacción de haber conseguido desarrollar una herramienta completa que pueda ser utilizada por varias personas sin olvidar la gran cantidad de aspectos que se han aprendido durante todo el proceso.

Trabajo Futuro

La realización de este proyecto de fin de carrera propone una nueva línea de trabajo en el desarrollo de algoritmos de flujo óptico y análisis de resultados. Utiliza metodologías y estrategias ya existentes y las integra en una única herramienta para facilitar su gestión y administración.

Se propone a continuación una serie de aspectos que podrían ser desarrollados de cara a mejorar y completar la aplicación que ha sido implementada:

- **Paralelizar otros algoritmos**: resulta una mejora inmediata dado que el objetivo consiste en seguir investigando y mejorando la precisión de los algoritmos de flujo óptico.
- **Adaptar para plugins**: adaptar la aplicación, ahora rudimentaria, que sólo sirve para paso de parámetros, a una más compleja que sirva para testear muchos más algoritmos desarrollados por terceros suministrados a modo de plugins.
- **Mejorar la lectura de imágenes:** suministrar la posibilidad de leer más formatos de imagen a la aplicación.
- **Procesamiento distribuido**: Implementar una versión del algoritmo que utilice al menos los distintos núcleos de un procesador multicore.
- **Imágenes en color**: el método abordado en este proyecto utiliza imágenes en escala de grises. Ampliar el desarrollo del algoritmo mediante el empleo de la información provista por los tres canales de las imágenes sin duda mejoraran las estimaciones.

Bibliografía

[1] G. E. Moore, «http://www.intel.com,» 8 Abril 1965. [En línea]. Available: ftp://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf. [Último acceso: 22 Junio 2013].

[2] G. E. Moore, «Progress in digital integrated electronics,» IEDM Technical Digest, IEEE International Electron Devices Meeting.

[3] J. Pastor, «the Inquirer,» [En línea]. Available: http://www.theinquirer.es/2007/09/19/gordon_moore_mi_ley_dejara_de_cumplirse_dentro_de_10_o_15_anos.html. [Último acceso: 22 Junio 2013].

[4] «Top500.org,» [En línea]. Available: https://www.top500.org/news/china-tops-supercomputer-rankings-with-new-93-petaflop-machine/. [Último acceso: 22 Junio 2013].

[5] «Top500.org,» [En línea]. Available: http://top500.org/lists/2008/06/. [Último acceso: 22 Junio 2013].

[6] M. Amartino, «Denken Über,» [En línea]. Available: http://www.uberbin.net/archivos/rants/roadrunner-la-supercomputadora-mas-rapida-hace-4-anos-ya-es-obsoleta.php. [Último acceso: 16 Mayo 2013].

[7] «Raspberrypi.org,» [En línea]. Available: http://www.raspberrypi.org/archives/2008. [Último acceso: 23 Junio 2013].

[8] «University of Southampton,» Septiembre 2012. [En línea]. Available: http://www.southampton.ac.uk/~sjc/raspberrypi/Raspberry_Pi_supercomputer_11Sept2012.pdf. [Último acceso: 24 Junio 2013].

[9] S. Cox, «University of Southampton,» Septiembre 2012. [En línea]. Available: http://www.southampton.ac.uk/~sjc/raspberrypi/pi_pictures_files/raspberry%20pi%20supercomputer%206.jpg. [Último acceso: 23 Junio 2013].

[10] ITCCanarias, «Supercomputador del Gobierno de Canarias,» ItcCanarias, [En línea]. Available: http://atlante.itccanarias.org/index.php?option=com_content&view=article&id=53&Itemid=78. [Último acceso: 13 Mayo 2013].

[11] J. S. Leitner, «Notebookcheck,» 1 Marzo 2011. [En línea]. Available: http://www.notebookcheck.net/Review-Asus-N53SV-Notebook.43709.0.html. [Último acceso: 24 Junio 2013].

[12] «Notebookcheck.org,» [En línea]. Available: http://www.notebookcheck.net/NVIDIA-launches-GeForce-GT-540M-Laptop-GPU.42219.0.html. [Último acceso: 24 Junio 2013].

[13] M. C. Vision, «The Middlebury Computer Vision Pages,» [En línea]. Available: http://vision.middlebury.edu/flow/eval/results/results-e1.php. [Último acceso: 8 Julio 2013].

[14] M. C. Vision, «The Middlebury Computer Vision Pages,» [En línea]. Available: http://vision.middlebury.edu/flow/eval/results/results-a1.php. [Último acceso: 8 Julio 2013].

[15] T. M. C. Vision, «The Middlebury Computer Vision Pages,» [En línea].

Available: http://vision.middlebury.edu/flow/eval/results/results-i1.php. [Último acceso: 8 Julio 2013].

[16] M. C. Vision, «The Middlebury Computer Vision Pages,» [En línea]. Available: http://vision.middlebury.edu/flow/eval/results/results-n1.php. [Último acceso: 8 Juli 2013].

[17] Bertero, 1988.

[18] R. S. Pressman, Ingeniería del Software. Un enfoque práctico., McGraw-Hill España, 2002.

[19] L. Álvarez, «Dis.ulpgc,» 01 01 2012. [En línea]. Available: http://www2.dis.ulpgc.es/~lalvarez/teaching/aan/ami_bmp.h. [Último acceso: 29 10 2016].

[20] OpenCV, «Opencv.org,» [En línea]. Available: http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture. [Último acceso: 29 10 2016].

[21] Nvidia, «http://www.nvidia.com,» 2009. [En línea]. Available: http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf.

[22] Hennesy-Patterson, «Ley de Amdahl,» de *Arquitectura de computadores - Un enfoque cuantitativo*, McGrawHill, 2002, p. 9.

Anexo 1. Manual de usuario

Se presenta a continuación el Manual de Usuario del sistema desarrollado. Se muestran en cada punto del anexo imágenes de la aplicación con una explicación detallada de sus funcionalidades y herramientas disponibles.

Para poder comprender correctamente cada una las explicaciones y guías aportadas, se explican a continuación cada uno de los elementos de la interfaz.

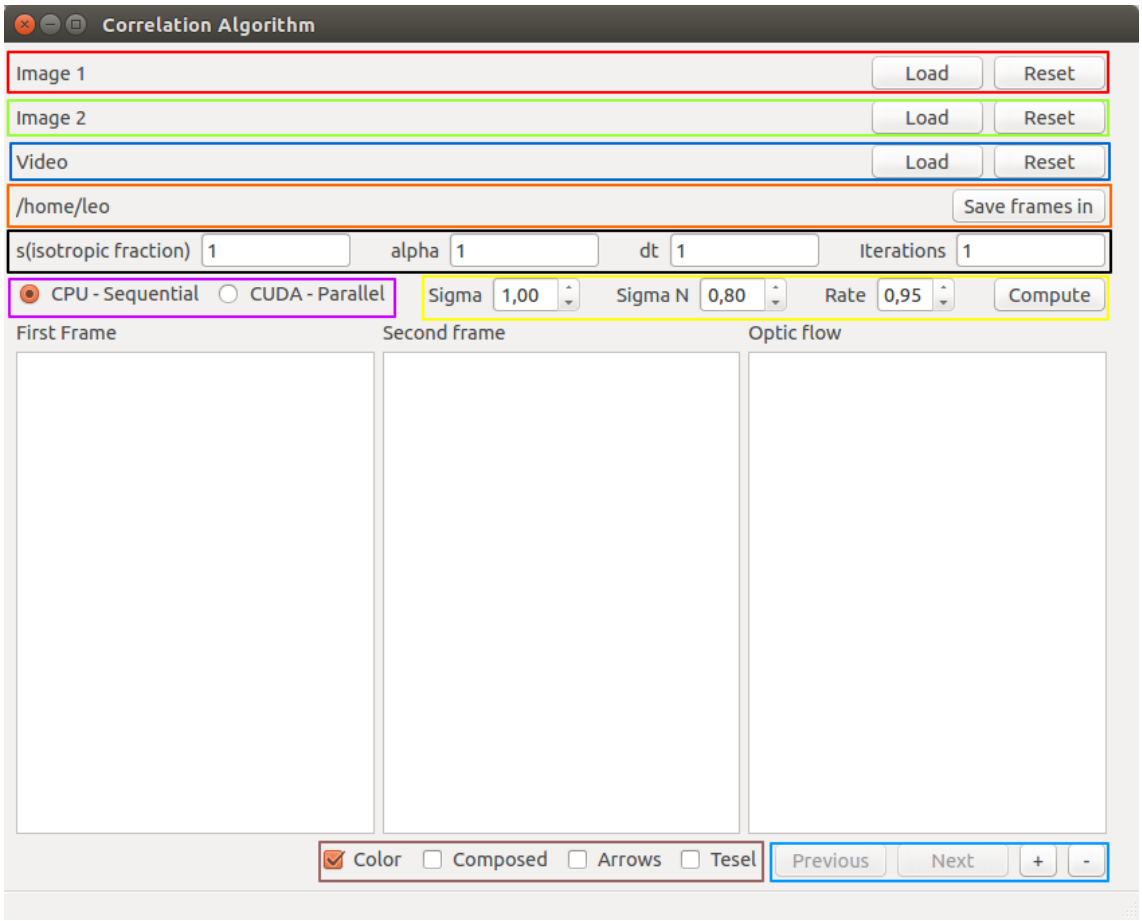


Imagen 37 - Manual de usuario

Como puede apreciarse en la imagen anterior, la sección comprendida por los primeros rectángulos, del rojo al naranja, es la sección de carga de ficheros; se ha habilitado para la gestión imágenes y resultados.

La segunda región capturada dentro del rectángulo negro es la zona de configuración donde se establecen los parámetros de entrada necesarios para el algoritmo.

La tercera zona, la zona de elección de dispositivo, es la que está delimitada por el recuadro de color rosa. A través de ella, el usuario indicará si usar la CPU o la GPU.

El botón de cálculo (compute) junto con la definición de las sigmas y la tasa de caida (decay rate) se encuentra en el área amarilla y es el detonante para qué si todo está bien configurado, comience la ejecución.

Dentro del rectángulo gris, pueden verse la sección de navegación donde podrán verse los resultados una vez se haya ejecutado el algoritmo juntamente con los botones de navegación (recuadro marrón) y los checkbox para alternar el tipo de imagen.

Carga de archivos

Para poder comenzar a calcular el flujo óptico, el usuario debe cargar al menos dos imágenes y hacer click en el botón “Compute”.

En la imagen anterior se muestra la pantalla qué se verá el usuario al cargar la aplicación. El primer paso qué debe efectuar es hacer click en el botón “Load” dentro del rectángulo rojo tras lo que el programa mostrará un diálogo de ficheros (File Dialog) típico de cualquier otra aplicación para abrir archivos. Seguidamente se dirigirá a la carpeta donde se halle dicho fichero y lo cargará vía doble click o seleccionándolo y haciendo click en aceptar.

De igual manera se procederá con la segunda imagen (recuadro verde) y, a su vez, con las secuencias de video (recuadro azul).

A continuación, una vez listas las imágenes, de forma opcional, el usuario podrá elegir en qué directorio guardar los resultados. Para ello, se procede de igual forma que para cargar un fichero, con la salvedad que esta vez sólo se llega hasta un directorio se selecciona y listo. El botón que debe usar es el que está dentro del recuadro naranja.

Cuando una imagen se carga de forma correcta, se previsualiza en las cajas de resultados delimitadas en nuestra imagen por el recuadro gris.

Configuración

El algoritmo necesita cuatro valores numéricos para poder funcionar. Su valor por omisión es uno. En caso de qué se produjese un error en el que se dejase vacío o se introdujese un valor de cualquier otro tipo, el programa volverá a asignar el último valor válido.

Para proceder dicha configuración, han de introducirse los valores de "s" y "alfa" qué son de tipo real comprendidos entre cero y uno. Posteriormente, "dt" que puede ser mayor que uno de la misma manera que el número de iteraciones aparte de positivos y mayores que cero.

Elección de dispositivo

La elección del entorno de proceso es, quizás, la operación más simple dentro del programa. Consiste en seleccionar uno de los dos botones de radio presentes dentro del rectángulo rosa.

El entorno elegido por defecto es la CPU dado que pudiera ocurrir que el equipo en el que se va a ejecutar el algoritmo no dispone de GPU, o los drivers no están instalados, o cualquier otra razón en cuyo caso el botón de radio correspondiente a la opción de GPU estará deshabilitada.

Ejecución

Una vez estén todos y cada uno de los parámetros correctamente establecidos, se procede a la ejecución simplemente haciendo click en el botón "Compute" que en nuestra imagen de ejemplo está dentro del rectángulo amarillo.

Dado que la única posibilidad de error donde la causa pudiese producto del usuario, si se intenta comenzar el cálculo del flujo sin una secuencia de video o dos frames de las mismas dimensiones y de tipo ".bmp", el programa mostrará una advertencia como la siguiente:

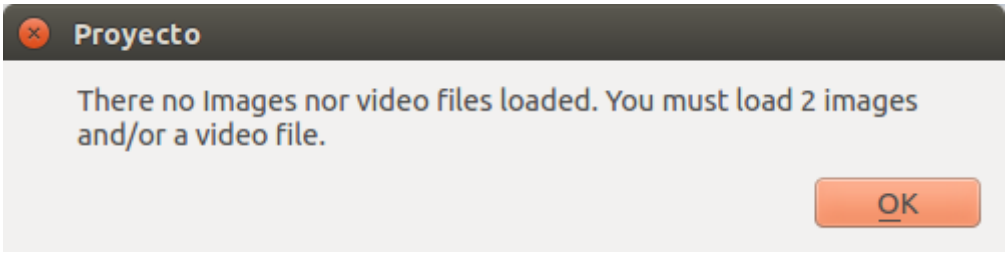


Imagen 38 - Message Box Error

Visualización de resultados

Dentro del recuadro gris, podemos ver tres cajas de imágenes donde, tras haber finalizado el cálculo del flujo óptico, se rellenarán (de derecha a izquierda) con la primera imagen, luego con la segunda y finalmente con el resultado obtenido por el algoritmo. Este tercer frame podrá alternarse entre dos posibles combinaciones:

1. La resultante de sumarle a la imagen 1 el desplazamiento obtenido
2. Imagen sintética creada como un mapa de colores a partir de la rueda de colores de referencia que puede verse desde el menú principal. Seguidamente se muestra cómo acceder:
    a. Utilizando el atajo de teclado "C" o mediante el menú "View"

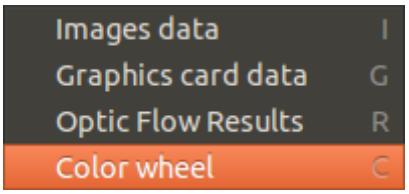


Imagen 39 - Menu contextual Color wheel

Imagen 40 - Paleta de colores

b. Visualización de resultados numéricos

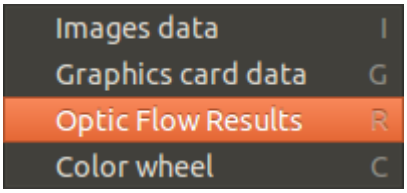


Imagen 41 - Menú contextual de resultados

Una vez ha terminado el cálculo, si se desea evaluar con precisión los resultados de flujo de un pixel en concreto, se puede acceder a él desde el menú principal (como se aprecia en la imagen precedente) o, una vez más, utilizando el atajo de teclado "R".

Visualización de las propiedades de las imágenes

Con el fin de tener un mayor entendimiento de las imágenes que se están utilizando en el programa existe una opción del menú que permite desplegar un cuadro con una tabla donde, siempre que haya cargada al menos una imagen, se podrán apreciar cada una de sus propiedades.

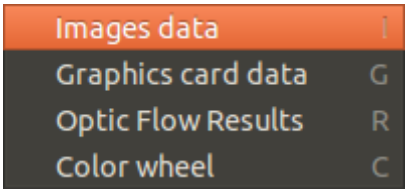


Imagen 42 - Menú contextual meta info de imágenes

El cuadro resultante se muestra a continuación:

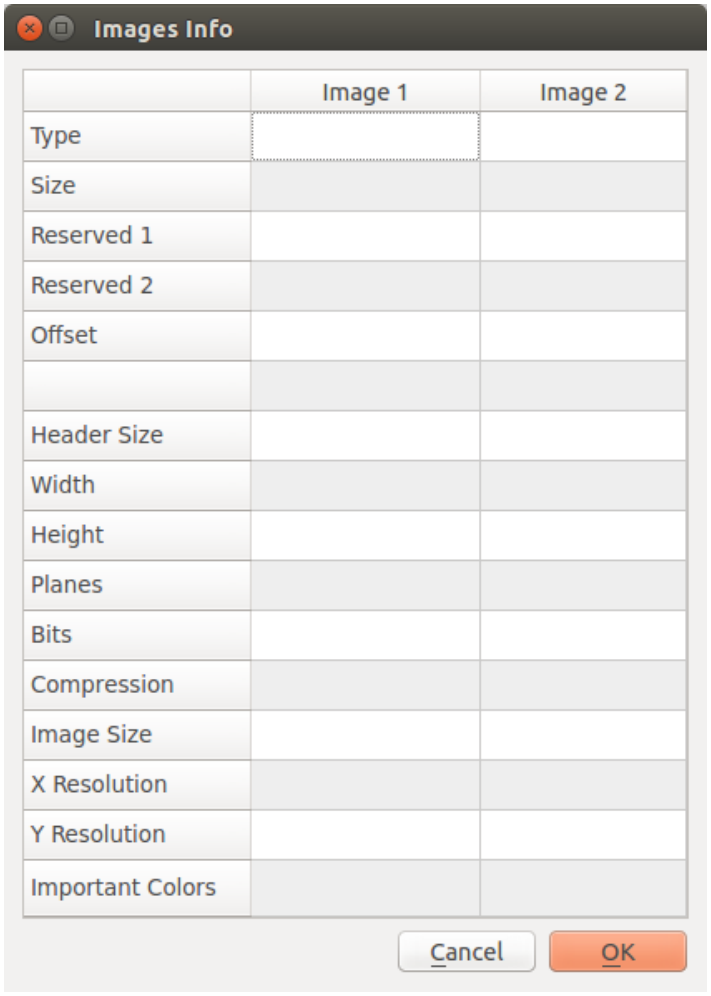


Imagen 43 - Tabla con meta información de imágenes

Visualización de las propiedades técnicas de las GPUS

Desde el menú View podemos acceder a Graphics card data

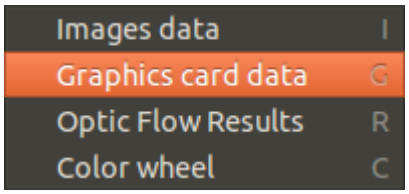


Imagen 44 - Información contextual de las tarjetas gráficas

Donde, si existe al menos una GPU en el sistema, podremos ver las propiedades técnicas de cada una para observar si es apropiada o no.

	Card 1	Description
Name	GeForce GT 540M	ASCII string identifying the device
totalGlobalMem	2147155968	total amount of global memory available on the device in bytes
sharedMemPerBlock	49152	maximum amount of shared memory available to a thread block in bytes; this amount is shared by all thread blocks simultaneo...
regsPerBlock	32768	maximum number of 32-bit registers available to a thread block; this number is shared by all thread blocks simultaneously resid...
warpSize	32	warp size in threads
memPitch	2147483647	maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through cudaMallocPitch()
maxThreadsPerBlock	1024	the maximum number of threads per block
maxThreadsDim[3]	1024, 1024, 64	contains the maximum size of each dimension of a block;
maxGridSize[3]	65535, 65535, 6...	contains the maximum size of each dimension of a grid;
clockRate	1344000	clock frequency in kilohertz;
totalConstMem	65536	total amount of constant memory available on the device in bytes;
major	2	major revision numbers defining the device's compute capability
minor	1	minor revision numbers defining the device's compute capability
textureAlignment	512	alignment requirement; texture base addresses that are aligned to textureAlignment bytes do not need an offset applied to text...
texturePitchAlignment	32	pitch alignment requirement for 2D texture references that are bound to pitched memory;
deviceOverlap	1	is 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not. Deprecated, use...
multiProcessorCount	2	the number of multiprocessors on the device;
kernelExecTimeoutEnabled	1	is 1 if there is a run time limit for kernels executed on the device, or 0 if not.
integrated	0	is 1 if the device is an integrated (motherboard) GPU and 0 if it is a discrete (card) component.
canMapHostMemory	1	is 1 if the device can map host memory into the CUDA address space for use with cudaHostAlloc()/cudaHostGetDevicePointer(),...
computeMode	0	is the compute mode that the device is currently in: Default, Exclusive, Prohibited, ExclusiveProcess
maxTexture1D	65536	is the maximum 1D texture size.

Cancel OK

Imagen 45 - Información contextual de la primera tarjeta gráfica

Finalización del programa

Cuando se haya terminado de realizar la labor pertinente, existen tres métodos para cerrar el programa:

1. Presionando la tecla escape
2. Utilizando la combinación de teclado tradicional Alt + F4
3. Utilizando el menú principal File y escogiendo la única opción existente: Exit como se aprecia en la imagen siguiente:

Imagen 46 - Exit

Anexo 2. Manual de instalación

Se presentan en este anexo los pasos a seguir para la instalación de los elementos necesarios para la ejecución del programa y posterior desarrollo del proyecto, si así se desea.

En primer lugar, resulta inevitable poseer una tarjeta gráfica externa o integrada en el equipo dado que si no será imposible ejecutar de manera óptima y completa el programa. De lo contrario, sólo se podrá utilizar la ejecución en CPU.

El programa es de por si independiente y totalmente funcional de forma autónoma por cuanto se proveen además las bibliotecas necesarias para su perfecto desempeño, es decir, tanto las librerías de Qt como las de CUDA, por lo que no requiere instalación de ninguna índole. No obstante, sigue siendo menester que esté presente en el sistema el driver de CUDA dado que las bibliotecas no pueden hacer nada por sí solas.

Instalación del material de desarrollo

Cuda

Requisitos

Para poder hacer uso de CUDA en un equipo es indispensable tener instalado:

- una GPU que soporte CUDA
- Entorno Linux (para ejecutar el programa)
- NVIDIA CUDA Toolkit (disponible en http://developer.nvidia.com/cuda-downloads)

Instalación en Ubuntu 14.10

Comprobación previa

Antes de comenzar el proceso hemos de cerciorarnos de que el siguiente comando produce error al ejecutarlo en el terminal:

```
$ nvcc –version
```

y que locate cuda no devuelve ningún resultado. Eso quiere decir que, efectivamente, CUDA no está instalado. De lo contrario, podrá apreciarse en la pantalla un resultado similar al siguiente (y por ende ignorar los pasos 2 al 4):

```
nvcc: NVIDIA ® Cuda compiler driver
Copyright © 2005-2015 NVIDIA Corporation
Built on Mon_Feb_16_22:59:02_CST_2015
Cuda compilation tools, release 7.0, V7.0.27
```

Repositorio CUDA

Descargar del repositorio de CUDA el paquete para Ubuntu 14.04 del sitio oficial de descargas CUDA download site y siga los pasos que se describen a continuación para instalarlos vía el terminal de comandos:

```
$ sudo dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb
$ sudo apt-get update
```

Toolkit CUDA

Seguidamente se ha de instalar el Toolkit de CUDA mediante apt-get.

```
$ sudo apt-get install cuda
```

A continuación el sistema operativo ha de ser reiniciado para posteriormente comprobar que todo ha funcionado como se espera.

Variables de entorno

Como penúltimo paso, deben asentarse las siguientes instrucciones en el fichero .bashrc para que la configuración de CUDA se lleve a cabo de forma permanente:

```
export CUDA_HOME=/usr/local/cuda-7.5
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64
PATH=${CUDA_HOME}/bin:${PATH}
export PATH
```

CUDA SDK Samples

Finalmente debe copiarse los ejemplos de código del SDK a la carpeta principal (home). Para ello deben ejecutarse las siguientes instrucciones en la consola

```
$ cuda-install-samples-7.5.sh  ~
$ cd ~/NVIDIA_CUDA-7.5_Samples
$ cd 1_Utilities/deviceQuery
$ make
```

Con el objetivo de verificar qué todo ha ido con normalidad, puede ejecutar el programa deviceQuery dentro de los ejemplos de CUDA.

Qt

Primeramente , al igual que en el paso anterior, hemos de asegurarnos qué versión de Qt existe en el sistema (si la hay) mediante los siguientes comandos:

```
which qmake
qmake -v
```

La salida del primer comando informa donde se encuentra el ejecutable qmake. Si dicha salida es similar a bash: qmake: command not found, es posible que:

1. Qt no esté instalado (incluyendo las herramientas de desarrollo)
2. Esté instalado pero la variable PATH del sistema no la contemple.

Por el contrario, si qmake existe:

```
$ qmake -v
QMake version 2.01a
Using Qt version 4.8.6 in /usr/lib/x86_64-linux-gnu
```

podrá apreciarse información acerca de la versión instalada y entonces procederemos a comprobar si las demás herramientas están en el equipo:

```
$ which moc
/usr/bin/moc
```

```
$ which uic
/usr/bin/uic
$ which assistant
/usr/bin/assistant
$ which designer
/usr/bin/designer
```

Si estos ejecutables están presentes Qt está listo para empezar a usarse.

A continuación se describe la forma más inmediata (y gráfica de proceder a la instalación de Qt y sus herramientas)

Para instalar Qt se deben seguir los pasos que se describen en su página web: http://qt-project.org/doc/qt-4.8/installation.html, según la plataforma en la que se realice la instalación.

No obstante, el proceso de instalación es muy simple. Consiste en descargar el ejecutable apropiado desde la página web: http://qt-project.org/downloads#qt-lib y seguir los pasos indicados en el documento. De hecho, para este proyecto se descargó un ejecutable para Ubuntu (.run) (http://download.qt.io/official_releases/online_installers/qt-unified-linux-x64-online.run) que provee de un entorno gráfico guiando paso a paso la instalación del mismo.

Configuración de Eclipse

Es necesario además un plugin para Eclipse qué brinda tres cajas de herramientas para compilar código fuente CUDA (NVIDIA) y/o Qt y además provee el generador de cabeceras H a partir de ficheros ui generados por Qt resultantes del diseño de la interfaz.

Han de seguirse los siguientes pasos:

1. Ejecutar Eclipse.
2. Ir a Ayuda → Instalar nuevo Software y añadir la siguiente URL: http://www.ai3.uni-bayreuth.de/software/eclipsecudaqt/updates/
3. Instalar.

Instalación de OpenCV

Finalmente sólo resta instalar OpenCV que es necesario para poder abrir las secuencias de video.

Los pasos para la instalación son muy sencillos dado que puede ser instalado vía terminal o aplicación gráfica. A continuación se listan los únicos tres comandos necesarios para hacerlo vía consola:

```
$ sudo apt-get install build-essential

$ sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev
libavformat-dev libswscale-dev

$ sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev
libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev.
```

Anexo 3. Modos de ejecución

Lo que hace posible posible que el programa se pueda ejecutar tanto vía consola como en entorno gráfico es qué es capaz de reconocer el modo de inicio con el que el usuario lo ha arrancado. Estos dos posibles modos son (1) doble click y (2) vía línea de comandos.

A continuación podemos ver en la imagen, que en la línea señalada por el recuadro rojo la función **isatty** que es la que dirime si la iniciación proviene de la terminal o del ratón. Esta función devuelve 1 si el descriptor de fichero abierto que se le pasa por parámetro se refiere a un terminal; de lo contrario devuelve 0. Por lo tanto, como se puede apreciar debajo, cuando devuelva 0 significa que debe ejecutarse el entorno gráfico. En otro caso, en los if anteriores el programa habrá realizado el procesamiento apropiado de los parámetros leídos en la consola por medio del "struct option".

```
    if (optind < argc) {
        printf ("non-option ARGV-elements: ");
        while (optind < argc)
            printf ("%s ", argv[optind++]);
        printf ("\n");
    }
    else if (optind == 1 && optind == argc) printUsage();

    // at least Image1 and Image2
    if (image1 and image2)
    {
        Application App(image1, image2, lambda, dt, cte, niter); // main application
        App.computeOpticFlow(version == 1); // true == CPU; false == CUDA
    }
    // at least video
    else if (video)
    {
        Application App(video, lambda, dt, cte, niter); // main application
        App.computeOpticFlow(version == 1); // true == CPU; false == CUDA
    }
    else if (!isatty(fileno(stdin))) {
        QApplication app(argc, argv);

        mainWindow mW;
        mW.show();

        return app.exec();
    }
```

Imagen 47 - Stdin, doble click ejecutable

Procesado de opciones

Para procesar las opciones qué pueda el usuario introducir por consola, dado que el análisis léxico y sintáctico de cualquier gramática siempre se complica se ha optado por usar las herramientas provistas por el sistema. En nuestro caso getopt_long que se encarga precisamente de esto empleando como argumentos argc y argv que son los parámetros de entrada de la función main. Cualquier elemento de argv que empieza con '-' (y no es exactamente "-" o "--") es un elemento de entrada, es decir, una opción.

A su vez, brinda la flexibilidad de poder introducir opciones de esta forma

--arg=param o de esta otra **--arg param**

Las opciones se configuran usando el struct option que posee la siguiente estructura:

```
struct option {
  const char *name;
  int         has_arg;
  int        *flag;
  int         val;
};
```

Name: es el nombre de la opción
has_arg: puede adoptar tres valores:

- **no_argument** (o 0) si la el parámetro no admite argumentos;
- **required_argument** (o 1) si la opción necesita de forma obligada un argumento y finalmente
- **optional_argument** (o 2) si, como se infiere de su nombre, el argumento es opcional.

Flag: especifica la manera en la que se devuelven los resultados
Val: es el valor qué se carga en la variable a la que apunta flag

```
int c;
int version    = 1,  niter = 100;
float lambda   = 1., cte   = 1., dt = 1.;
char * image1  = 0,
     * image2  = 0,
     * results = 0,
     * video   = 0;

static struct option long_options[] = {
    { "image1",  required_argument, 0,  'i' },
    { "image2",  required_argument, 0,  'j' },
    { "video",   required_argument, 0,  'v' },
    { "results", required_argument, 0,  'r' },
    { "lambda",  required_argument, 0,  'l' },
    { "cte",     required_argument, 0,  'c' },
    { "dt",      required_argument, 0,  'd' },
    { "niter",   required_argument, 0,  'n' },
    { "seq",     no_argument,       0,  's' },
    { "par",     no_argument,       0,  'p' },
    { NULL,      0,                 NULL, 0 }
};
```

Imagen 48 - Procesamiento de opciones

Apéndice. Detalles sobre la implementación

Guía de compilación

Para compilar el proyecto es necesario que la versión del driver sea compatible o superior a la versión del Toolkit de CUDA. Para la versión 2.3 se debe disponer como mínimo del driver 190.x.

El primer paso es constatar que eclipse se encuentra instalado en el sistema; si no lo está, puede descargarse de https://eclipse.org/downloads/packages/eclipse-ide-cc-developers/heliossr2. Para este proyecto se ha empleado la versión Helios de eclipse:

Version: Helios Service Release 2 Build id: 20110218-0911

Posteriormente, se ha de instalar http://www.ai3.uni-bayreuth.de/software/eclipsecudaqt/

Para instalar este plugin es necesario añadir el siguiente sitio de actualización para eclipse en el gestor de actualizaciones mediante el menú Help → Install New Software:

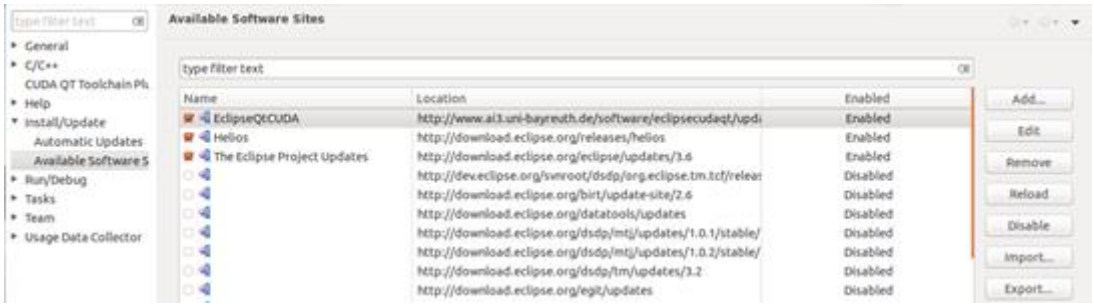


Imagen 49 - Propiedades del proyecto

La cadena de herramientas se aplica al proyecto. Las diferentes configuraciones, como la especificación de comandos puede hacerse dentro de la página de configuración:

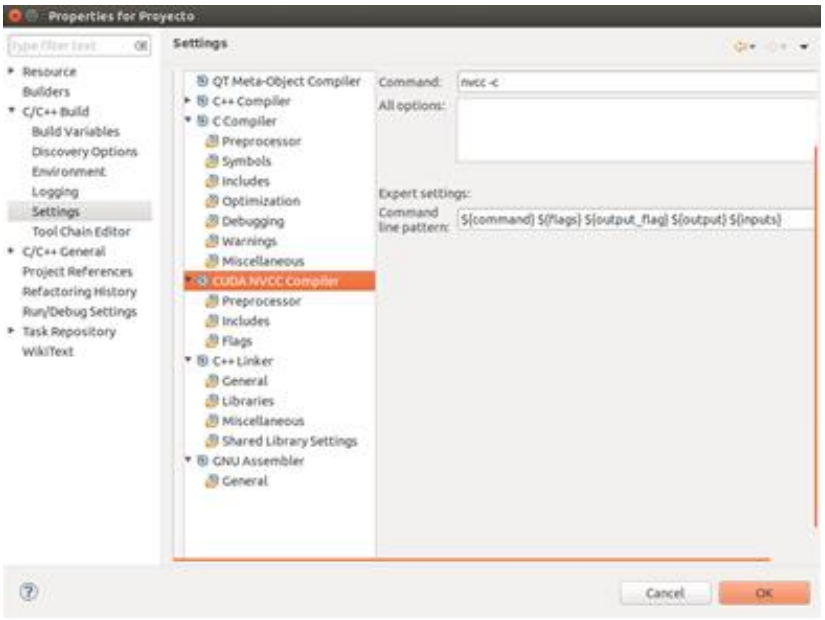


Imagen 50 - CUDA NVCC

La siguiente captura de pantalla muestra la posibilidad de compilar archivos ui utilizando el menú contextual en el Explorador de proyectos:

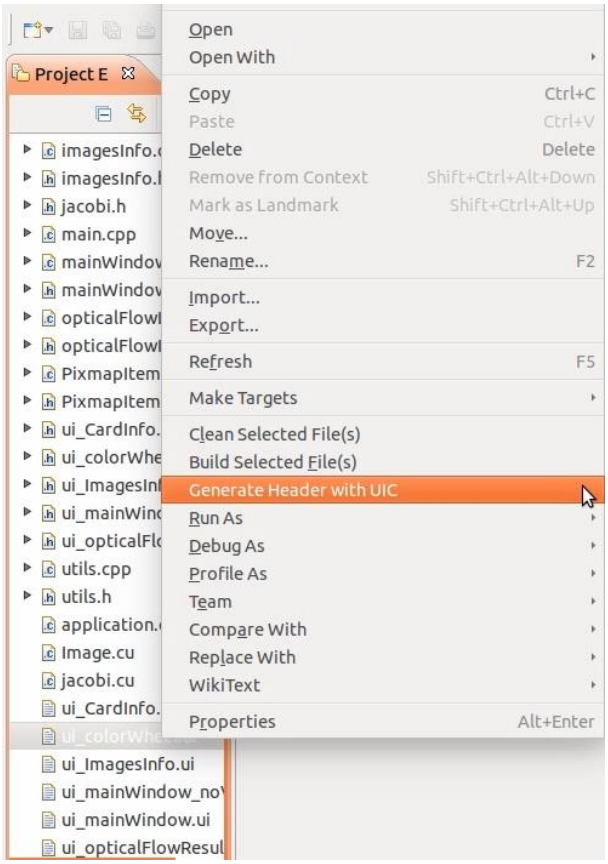


Imagen 51 - Compilar archivo .ui

Finalmente, importaremos el proyecto a Eclipse a través del menú archivo, import

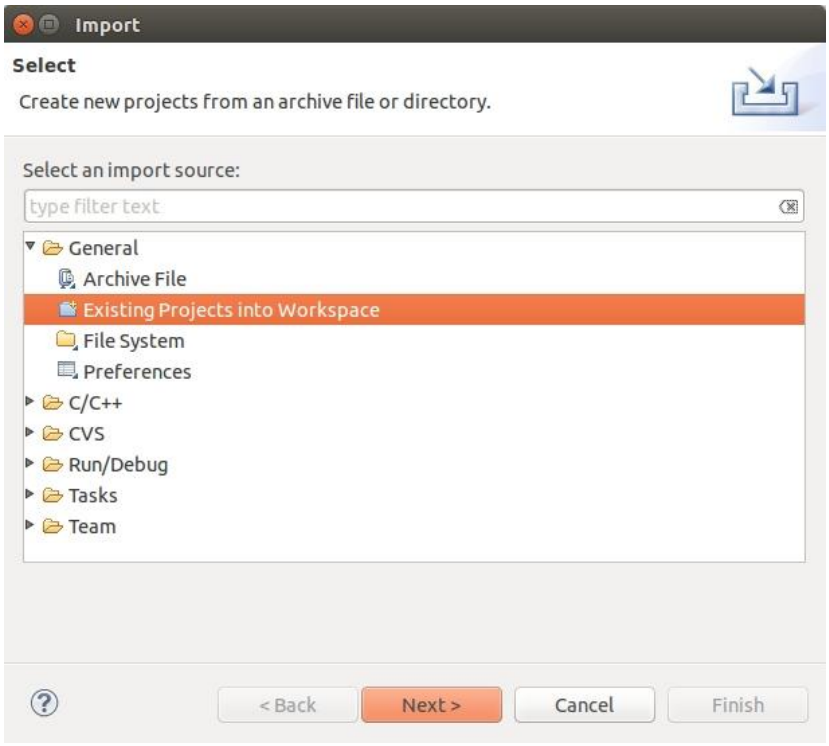


Imagen 52 - Importar proyecto

y seleccionaremos la opción "*Existing projects into workspace*", donde añadiremos el directorio del proyecto. Seguidamente nos aseguraremos de que las siguientes opciones están configuradas de la siguiente forma:

Toolchain editor:

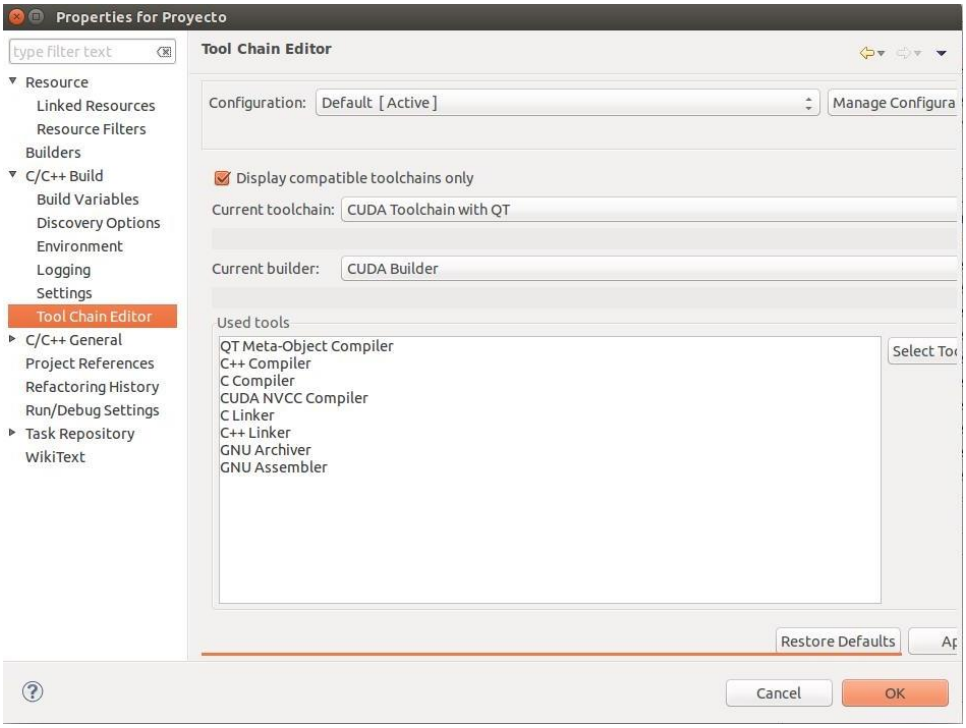


Imagen 53 - Toolchain editor eclipse

Para configurar las herramientas de compilación del proyecto, se ha de acceder a las propiedades del mismo y desplegar la segunda entrada del menú "*C/C++ Build*" y en la subsección "*ToolChain Editor*" seleccionar "*Cuda Toolchain with QT*" en la entrada de "*Current toolchain*" y "*Cuda Builder*" en la sección de "*Current builder*". Esta configuración permitirá que los archivos de C/C++ (.c, .cpp, .h, .hxx) sean compilados mediante GCC o G++ a la vez que los archivos cuda (.cu, .cuh) sean compilados mediante NVCC y posteriormente enlazados.

QT MOC

El compilador de meta objetos, moc (Meta-Object Compiler), de QT es el programa que se ocupa de manejar las extensiones de C++ de QT. Lo que hace es leer las cabeceras de un fichero C++ y si encuentra una o más declaraciones

que contengan la macro *Q_OBJECT*, produce un fichero cabecera (.h) de C++ que contiene el código meta objeto para esas clases. Entre otras cosas, dicho código meta objeto es necesario para poner en marcha el mecanismo de señales y ranuras (signals y slots) que es el responsable de gestionar la comunicación con el usuario a través de la interfaz gráfica (clicks, etc.).

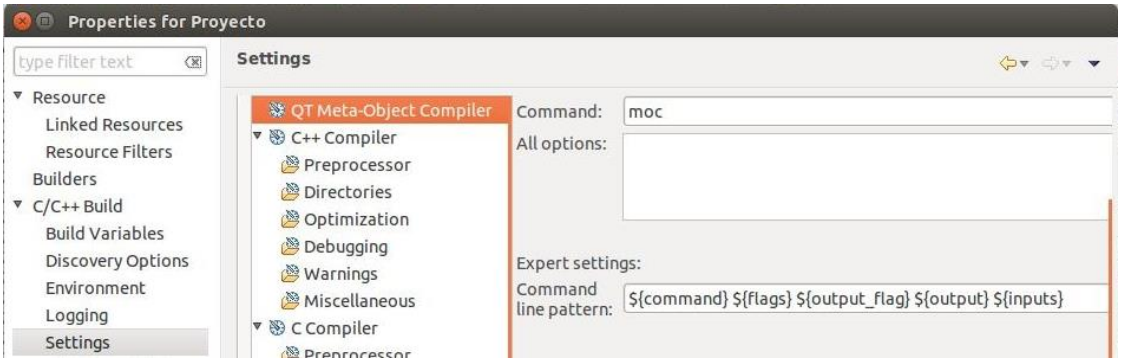


Imagen 54 - QT MOC (Meta object compiler)

El compilador de C++ ha de ser G++.

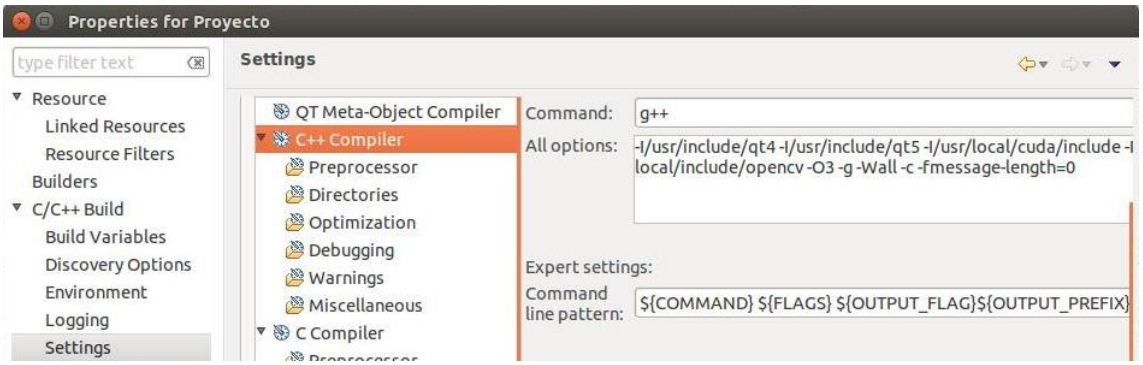


Imagen 55 - Compilador de C++ - Librerías

A su vez, puede apreciarse que dentro del recuadro que dice "*All options*" hay presente una lista de opciones del tipo "*-I*" seguido de una ruta absoluta. Son las bibliotecas necesarias (QT, Cuda y OpenCV) para la compilación del proyecto que se añadirán en el apartado C++ Linker.

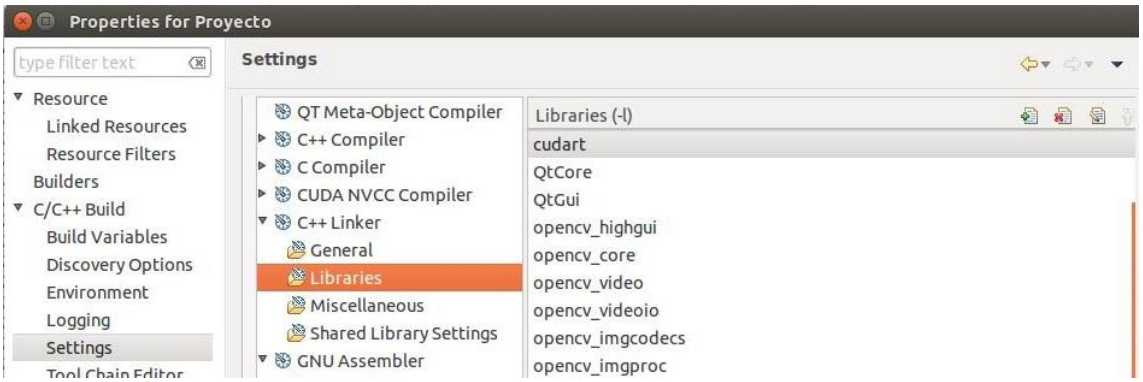


Imagen 56 - Bibliotecas

Finalmente los directorios con los ficheros cabecera de cuda, Qt y OpenCV han de ser añadidos también en la configuración. Se suelen encontrar en "*/usr/include/*" o "*/usr/local*"

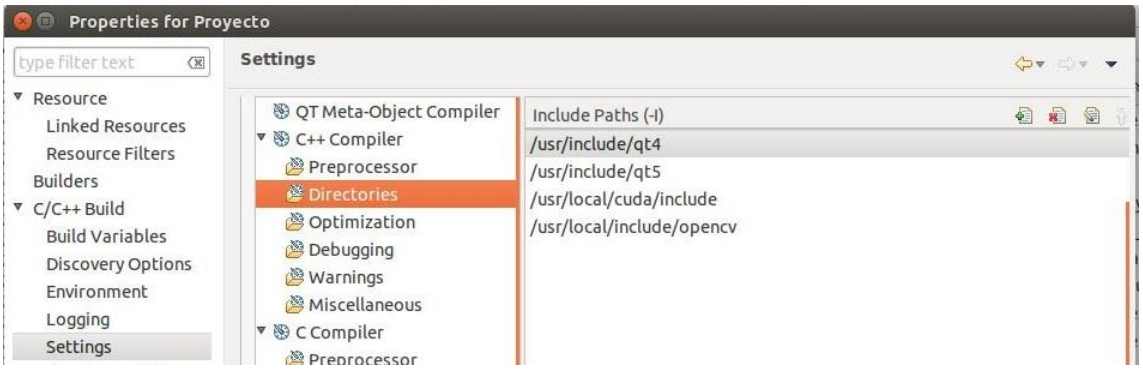


Imagen 57 - Directorios include