

# **Software de obtención y procesamiento de datos a tiempo real con CUDA para sistema de interferometría**

## *Memoria del Proyecto*

Proyecto de Fin de Carrera  
INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS



**VNiVERSiDAD  
E SALAMANCA**

Septiembre de 2012

Autor  
**Álvaro Sánchez González**

Tutor  
**Guillermo González Talaván**

Cotutor  
**Francisco Valle Brozas**



## Contenido

<b>1. Introducción .....</b>	<b>9</b>
1.1. <i>Estructura de la documentación .....</i>	11
<b>2. Objetivos del proyecto .....</b>	<b>15</b>
2.1. <i>Motivación .....</i>	15
2.2. <i>Objetivos funcionales y no funcionales.....</i>	16
2.3. <i>Objetivos técnicos.....</i>	18
<b>3. Conceptos teóricos .....</b>	<b>19</b>
3.1. <i>La luz como onda: Interferometría.....</i>	19
3.1.1. Interferómetro de Mach-Zehnder .....	23
3.2. <i>Transformada de Fourier Bidimensional Compleja.....</i>	25
3.2.1. Transformada de Fourier Unidimensional .....	25
3.2.2. Transformada de Fourier Espacial Bidimensional .....	26
3.2.3. Transformada de Fourier Rápida (FFT) .....	27
3.3. <i>Regresión lineal: Ajuste a mínimos cuadrados .....</i>	31
3.4. <i>Pre-procesado de una imagen con una referencia.....</i>	32
3.4.1. Obtención de la fase .....	33
3.4.2. Haciendo la fase continua .....	36
3.4.3. Restando la referencia .....	38
3.5. <i>Tratamiento posterior de la imagen de la fase .....</i>	39
3.5.1. Inversión de la imagen .....	40
3.5.2. Eliminación de la inclinación horizontal.....	40
3.5.3. Eliminación de la inclinación vertical.....	41
3.5.4. Fijado del nivel base.....	42

3.5.5.	Añadido de un valor.....	43
3.5.6.	Multiplicación por un factor.....	43
3.5.7.	Simetrización .....	44
3.5.8.	Inversión de Abel .....	44
3.5.9.	Extracción de un rectángulo.....	47
3.5.10.	Extracción de una línea .....	47
3.6.	<i>Experimento en el laboratorio</i> .....	48
<b>4.</b>	<b>Técnicas y herramientas .....</b>	<b>55</b>
4.1.	<i>Programación orientada a objetos: C++</i> .....	55
4.2.	<i>Qt y Microsoft Visual C++ 2008 Express Edition</i> .....	56
4.3.	<i>Bibliotecas de gráficos</i> .....	57
4.3.1.	Biblioteca de gráficos QwtPlot .....	57
4.3.2.	Biblioteca de gráficos QwtPlot3D .....	58
4.4.	<i>Biblioteca de compresión QuaZIP</i> .....	58
4.5.	<i>API μEye</i> .....	59
4.6.	<i>Procesamiento en paralelo: CUDA</i> .....	59
4.6.1.	Arquitectura .....	60
4.6.2.	Programación .....	63
4.6.3.	Ejemplo de uso .....	65
4.6.4.	Biblioteca cuFFT .....	67
4.6.5.	Comparativa de velocidad: CPU vs CUDA .....	67
4.7.	<i>Archivos INI</i> .....	69
4.8.	<i>XML</i> .....	70
4.9.	<i>MPlayer: MEncoder</i> .....	71
4.10.	<i>ImageMagick: Converter</i> .....	71
4.11.	<i>RealWorld Icon Editor</i> .....	72
4.12.	<i>Install Creator Pro</i> .....	72
4.13.	<i>Herramientas para la documentación</i> .....	73
<b>5.</b>	<b>Aspectos relevantes del desarrollo del proyecto</b> .....	<b>75</b>

5.1.	<i>Planificación</i> .....	75
5.2.	<i>Arquitectura utilizada</i> .....	77
5.2.1.	Modelo-Vista-Controlador modificado.....	78
5.2.2.	Vista de capas: .....	79
5.3.	<i>Descripción de la interfaz gráfica</i> .....	81
5.3.1.	Tipos de gráficos y sus tipos de datos.....	81
5.3.2.	Ventanas y diálogos disponibles.....	85
5.4.	<i>Algunos diagramas de clases de la vista estática</i> .....	95
5.4.1.	Estructura interna del programa .....	95
5.4.2.	Diagramas de clases con las estructuras y parámetros de cálculo .....	96
5.4.3.	Ejemplo de diagrama de clases para una ventana .....	97
5.5.	<i>Conviviendo CUDA y CPU: Factoría Abstracta</i> .....	98
5.6.	<i>Optimización de código para obtener rendimiento</i> .....	103
5.7.	<i>Paralelización de algoritmos para CUDA</i> .....	104
5.7.1.	Escaneo de la fase.....	105
5.7.2.	Eliminación de la inclinación horizontal/vertical.....	106
5.7.3.	Fijado del nivel base.....	107
5.7.4.	Simetrización .....	107
5.7.5.	Inversión de Abel .....	108
5.7.6.	Extracción de una línea .....	109
5.7.7.	Resto de algoritmos .....	110
5.8.	<i>Secuencia de procesamiento</i> .....	110
5.8.1.	Obtención de la fase de referencia.....	111
5.8.2.	Pre-procesado de una imagen.....	112
5.8.3.	Tratamiento posterior de la imagen de fase.....	113
5.9.	<i>Política de cálculo</i> .....	114
5.10.	<i>Cámara y simulador de cámara: Factoría Abstracta</i> .....	115
5.11.	<i>Toma de imágenes de forma eficaz</i> .....	118
5.12.	<i>Modo “en vivo”:</i> Mejora de la experiencia del usuario .....	122

5.13.	<i>Guardado en ficheros</i> .....	124
5.13.1.	Máscaras, parámetros y opciones de visualización.....	124
5.13.2.	Lista de algoritmos .....	126
5.13.3.	Lista de imágenes .....	128
5.13.4.	Preferencias del programa y parámetros de la cámara .....	130
5.13.5.	Guardado de proyecto .....	131
5.13.6.	Exportado de datos.....	132
5.13.7.	Exportado de vídeos y animaciones .....	135
5.14.	<i>Detalles para un uso más cómodo</i> .....	135
<b>6.</b>	<b>Trabajos relacionados</b> .....	<b>139</b>
6.1.	<i>Caracterización y optimización de un jet de gas</i> .....	139
6.2.	<i>IDEA: Interferometrical Data Evaluation Algorithms</i> .....	140
<b>7.</b>	<b>Conclusiones</b> .....	<b>143</b>
<b>8.</b>	<b>Líneas de trabajo futuras</b> .....	<b>147</b>
8.1.	<i>Nuevas cámaras</i> .....	147
8.2.	<i>Nuevas formas de exportar resultados</i> .....	147
8.3.	<i>Nuevos algoritmos</i> .....	148
8.4.	<i>Nuevos modos de cálculo</i> .....	148
<b>9.</b>	<b>Agradecimientos</b> .....	<b>151</b>
<b>10.</b>	<b>Referencias y bibliografía</b> .....	<b>153</b>

## Índice de Figuras

Figura 1. Imagen de un jet de gas, generada con la aplicación.....	9
Figura 2. Logo de la aplicación.....	10
Figura 3. Experimento de la doble rendija de Young.....	20
Figura 4. Onda electromagnética monocromática.....	20
Figura 5. Leyes de Maxwell en el vacío .....	21
Figura 6. Leyes de Maxwell en el vacío sin fuentes .....	21
Figura 7. Campo eléctrico propagándose .....	22
Figura 8. Interferencia constructiva y destructiva.....	23
Figura 9. Interferencia de dos frentes de onda .....	24
Figura 10. Fórmula de cálculo de la transformada de Fourier .....	25
Figura 11. Fórmula de cálculo de la transformada Fourier inversa.....	25
Figura 12. Ejemplos de módulos de funciones y sus transformadas.....	26
Figura 13. Ecuación de la transformada de Fourier bidimensional.....	26
Figura 14. Ecuación de la transformada de Fourier bidimensional inversa.....	27
Figura 15. Ejemplo de transformada de Fourier de una imagen .....	27
Figura 16. Ecuación de la transformada de Fourier unidimensional discreta .....	28
Figura 17. Ecuación de la transformada de Fourier bidimensional discreta .....	28
Figura 18. Reordenación de la transformada de Fourier tras el algoritmo FFT .....	30
Figura 19. Regresión lineal .....	31
Figura 20. Ecuaciones de la regresión lineal .....	32
Figura 21. Imagen de la interferencia adquirida por la cámara .....	33
Figura 22. Imagen ya adecuada para el pre-procesado.....	34
Figura 23. Transformada de Fourier de la imagen .....	34

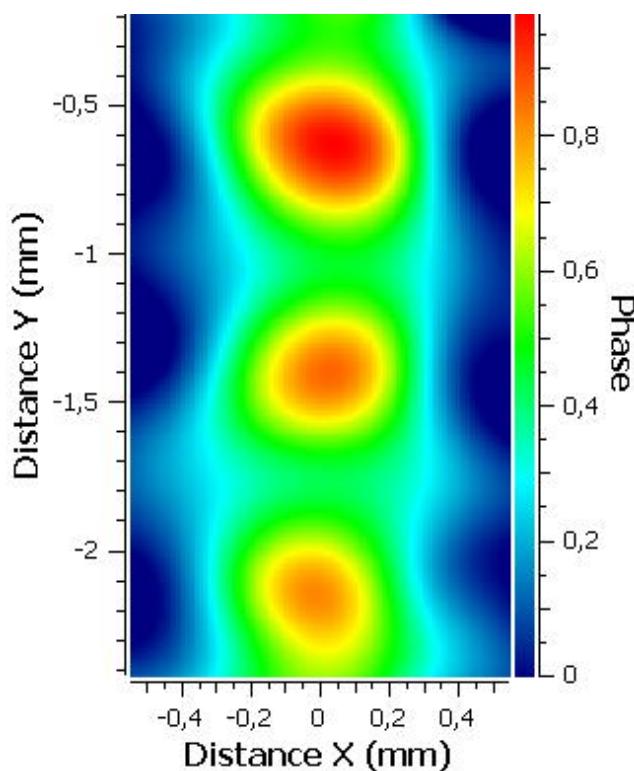
Figura 24. Transformada de Fourier con la máscara en el término de la fase .....	35
Figura 25. Fase tras realizar la transformada inversa .....	36
Figura 26. Fase en la zona de interés .....	36
Figura 27. Imagen de la fase continua tras el escaneado .....	38
Figura 28. Primera imagen del jet de gas .....	39
Figura 29. Operación de inversión, antes y después .....	40
Figura 30. Eliminación de la inclinación horizontal, antes y después .....	41
Figura 31. Eliminación de la inclinación vertical, antes y después .....	42
Figura 32. Operación de fijado de nivel base, antes y después .....	42
Figura 33. Operación de añadir un valor, antes y después .....	43
Figura 34. Operación de multiplicar por factor, antes y después .....	43
Figura 35. Operación de simetrización, antes y después .....	44
Figura 36. Láser al atravesar el jet de gas .....	45
Figura 37. Operación de inversión de Abel .....	46
Figura 38. Operación de extracción de rectángulo, antes y después .....	47
Figura 39. Diálogo de selección de recta .....	48
Figura 40. Esquema del montaje en el laboratorio .....	49
Figura 41. Imagen del montaje en el laboratorio .....	49
Figura 42. Imagen del cambio de fase sin procesar al pasar por el jet de gas .....	50
Figura 43. Imagen del mismo cambio de fase, procesado, y simetrizado .....	51
Figura 44. Valores del índice de refracción a cada altura y distancia al centro .....	51
Figura 45. Reconstrucción del jet de gas .....	52
Figura 46. Diamantes de choque en un motor a reacción .....	52
Figura 47. Diamantes de choque en un caza F-16 .....	53
Figura 48. Logo de C++ .....	55
Figura 49. Logo de Qt .....	56
Figura 50. Gráficos Graph1D y SpectrogramPlot respectivamente .....	57
Figura 51. Surface3DPlot y ParametricSurface3DPlot respectivamente .....	58
Figura 52. Logo de uEye® .....	59

Figura 53. Logo de NVIDIA CUDA® .....	59
Figura 54. Arquitectura de CUDA junto con la CPU .....	61
Figura 55. Distribución del procesamiento con CUDA.....	62
Figura 56. Distribución del trabajo para el procesado pixel a pixel .....	65
Figura 57. Comparativa entre CPU y CUDA, en escala natural y logarítmica .....	68
Figura 58. Logo de MPlayer .....	71
Figura 59. Logo de ImageMagick .....	72
Figura 60. Patrón Modelo-Vista-Controlador modificado .....	78
Figura 61. Ejemplo de MVC modificado.....	79
Figura 62. Diagrama de capas del sistema .....	80
Figura 63. Diagrama de clases de los objetos de resultados .....	81
Figura 64. Diagrama con las clases gráficas .....	82
Figura 65. Imagen del gráfico Graph1D .....	83
Figura 66. Imagen del gráfico SpectrogramPlot .....	83
Figura 67. Imagen del gráfico Surface3DPlot .....	84
Figura 68. Imagen del gráfico ParametricSurface3DPlot .....	84
Figura 69. Imagen de la ventana principal .....	85
Figura 70. Barra de herramientas .....	86
Figura 71. Lista de algoritmos.....	86
Figura 72. Zona central de gráficos.....	87
Figura 73. Lista de imágenes .....	87
Figura 74. Barra de estado .....	87
Figura 75. Selección de la imagen de la referencia .....	88
Figura 76. Fijando tamaño.....	88
Figura 77. Fijando la máscara y el origen de coordenadas .....	89
Figura 78. Fijando la máscara de Fourier .....	90
Figura 79. Resultado para la fase de la referencia .....	90
Figura 80. Diálogo de selección de recta .....	91
Figura 81. Otros cuadros de diálogos de introducción de parámetros .....	91

Figura 82. Ventana del modo “en vivo” .....	92
Figura 83. Cuadro de diálogo de configuración de la cámara .....	93
Figura 84. Diálogo de selección de modo .....	94
Figura 85. Diálogo de preferencias del programa .....	94
Figura 86. Estructura interna del programa a primer nivel .....	95
Figura 87. Controlador principal y sus relaciones a segundo nivel .....	96
Figura 88. Controlador de imágenes y sus relaciones a segundo nivel .....	97
Figura 89. Diagrama de clases de la ventana del modo “en vivo” .....	98
Figura 90. Factorías para los cálculos .....	100
Figura 91. Escaneado de fase: Paso 1 .....	105
Figura 92. Escaneado de fase: Paso 2 .....	106
Figura 93. Simetrización: Paso 2 .....	108
Figura 94. Diagrama de Secuencia: Procesado de la referencia .....	111
Figura 95. Diagrama de Secuencia: Pre-procesado de una imagen .....	112
Figura 96. Diagrama de Secuencia: Aplicación de algoritmos .....	113
Figura 97. Diagrama de Secuencia: Gráficos en la ventana principal.....	115
Figura 98. Clase de cámara abstracta .....	116
Figura 99. Abstracción y factorías de la cámara .....	117
Figura 100. Controlador de cámara y sus relaciones a segundo nivel .....	118
Figura 101. Diagrama de Secuencia: Hilo de toma de imágenes .....	121
Figura 102. Diagrama de Secuencia: Procesado de imagen en modo “en vivo” .....	123
Figura 103. Archivo de proyecto.....	132
Figura 104. Gráfico generado cargando los datos en MATLAB .....	134
Figura 105. Portada TFM Francisco Valle Brozas.....	139
Figura 106. Acerca de IDEA .....	140
Figura 107. Imagen de la aplicación IDEA .....	141

## 1. Introducción

El proyecto surge en el entorno del Área de Óptica[1] del Departamento de Física Aplicada[2] de la Universidad de Salamanca[3], motivado por la necesidad de un software que permitiera y agilizara la caracterización de un jet de gas para la aceleración de electrones con láser, tema sobre el que versa el proyecto de fin de máster del cotutor, Francisco Valle Brozas, persona clave de referencia en lo que a las cuestiones teóricas se refiere durante todo el proyecto.

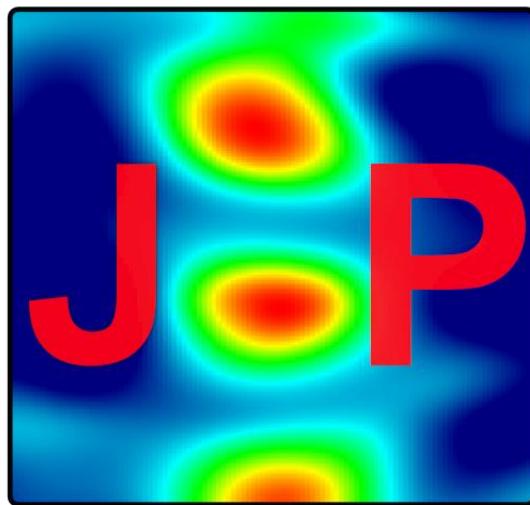


*Figura 1. Imagen de un jet de gas, generada con la aplicación*

Un “jet de gas” consiste en una corriente de gas generada artificialmente con un pistón y una válvula que controle el flujo, y “caracterizar un jet de gas” consiste en ser capaz de determinar, tanto de forma cualitativa, como de forma cuantitativa, cual es la densidad o índice de refracción del gas en cada punto del espacio.

Realizar esa tarea no es fácil. Por una parte, el jet de gas es transparente, por lo que a simple vista es imposible de observar, ni siquiera de forma cualitativa. Por otra parte, este proceso de caracterización se debe realizar de la forma menos intrusiva posible

para el jet de gas, dado que, por ejemplo, la introducción de una sonda cercana al jet, afectaría sobremanera a la distribución del propio jet, y además, haría imposible el objetivo final: realizar experimentos con el jet de gas al mismo tiempo que se está caracterizando, pudiendo así controlar al detalle los parámetros de interés del experimento.



*Figura 2. Logo de la aplicación*

La solución a esto es el uso de un sistema de interferometría con láser que permite, mediante el uso de un láser pulsado, la generación de un patrón de interferencia del que se puede extraer la información de la densidad del jet, siguiendo un proceso de obtención y filtrado de la imagen que se detallará en los próximos capítulos.

Sin embargo ese proceso no es inmediato, ni conceptual, ni computacionalmente hablando. Para el trabajo a tiempo real, requiere procesar imágenes de gran tamaño, a una tasa de, idealmente, 10 imágenes por segundo, lo que, en la práctica, resulta prácticamente imposible para una CPU normal. Esta es la clave del uso de CUDA en el proyecto. CUDA es una herramienta disponible en casi todas las tarjetas gráficas modernas de NVIDIA que permite realizar cálculos en paralelo en el multiprocesador de la propia tarjeta gráfica, la GPU, compilando código específico para ello. Para más información véase el capítulo 4.6.

Este “Software de obtención y procesamiento de datos a tiempo real con CUDA para sistema de interferometría”, como indica el nombre del proyecto, o simplemente “Jet Processing”, el nombre de la aplicación creada y que se utilizará de ahora en adelante, permite realizar todas las operaciones habitualmente necesarias para la caracterización de un jet de gas en un sistema de interferometría, o bien a tiempo real, con un ordenador montado sobre el sistema, o bien a posteriori, con imágenes tomadas con anterioridad durante el experimento, utilizando siempre que sea posible procesamiento en paralelo con la GPU para realizar los cálculos de forma más rápida.

Como último detalle, cabe destacar que aunque en el contexto del proyecto siempre se hable del procesado de la imagen de un jet de gas, la aplicación creada sirve para obtener la imagen de cualquier medio o material que se pueda atravesar con un láser, y no solamente un jet de gas.

### ***1.1. Estructura de la documentación***

La documentación que acompaña al proyecto, se ha generado siguiendo el documento “Proyecto de Fin de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación”, constando así de dos partes.

La primera parte consiste en el presente documento impreso: una descripción del proyecto. Este documento pretende recoger las características más importantes del proyecto, dando un contexto al proyecto, y permitiendo al lector aprender un poco más acerca de la teoría detrás del proyecto, así como de las herramientas y técnicas utilizadas, entrando sólo a detalles técnicos de diseño o de programación cuando son de suficiente interés, y omitiendo el resto para la segunda parte.

Los capítulos que se podrán encontrar en las siguientes páginas son:

- **Objetivos del proyecto:** resumiendo de forma sencilla y concisa los objetivos y requisitos para el software sobre el que trata el proyecto.
- **Conceptos teóricos:** donde se detallan todos los conceptos necesarios para la adecuada compresión del proyecto.

- **Técnicas y herramientas:** mostrando las herramientas utilizadas para alcanzar los objetivos del proyecto.
- **Aspectos relevantes del desarrollo del proyecto:** donde se resume el ciclo de vida que se ha seguido a lo largo del proyecto, así como los aspectos específicos de diseño más interesantes.
- **Trabajos relacionados:** donde se muestran otros trabajos que tratan sobre el mismo tema.
- **Conclusiones:** donde se realiza una breve reflexión sobre el desarrollo del proyecto y el producto final.
- **Líneas de trabajo futuras:** donde se proponen algunas vías de continuación para el proyecto.
- **Referencias y bibliografía:** reuniendo en un mismo lugar todas las fuentes que se mencionan a lo largo de la memoria.
- **Agradecimientos.**

La segunda parte del proyecto consiste en una serie de anexos técnicos proporcionados en soporte digital que consisten en los siguientes documentos:

- **Anexo I: Plan de Proyecto Software:** muestra la característica temporal del proyecto. Documento PDF de aproximadamente 25 páginas.
- **Anexo II: Especificación de Requisitos del Software:** recoge los objetivos, requisitos, y casos de uso, así como un modelo de análisis del proyecto. Documento PDF de aproximadamente 95 páginas.
- **Anexo III: Especificación de diseño:** detalla el modelo de diseño, incluyendo las técnicas utilizadas para la creación del software, de forma que el código creado siga una buena programación, modular, cohesivo, pero sin acoplamiento, y cumpliendo también los requisitos no funcionales. Documento PDF de aproximadamente 65 páginas.
- **Anexo IV: Documentación Técnica de Programación:** proporciona todos los detalles prácticos de puesta a punto del entorno de desarrollo, programación y compilación, así como la documentación del código realizado. Documento PDF

de aproximadamente 20 páginas complementado con una página web en HTML con la documentación del código.

- **Anexo V: Manuales de usuario:** muestra los manuales de configuración del equipo, instalación y de uso de la aplicación. Documento PDF de aproximadamente 70 páginas o web en HTML, disponible en español y en inglés.



## 2. Objetivos del proyecto

En lo que sigue se exponen los objetivos del proyecto.

Se comienza por los motivos que llevaron a la idea de la creación del proyecto, para pasar a los objetivos en sí: funcionales, no funcionales y técnicos.

Información mucho más detallada sobre los objetivos, y cómo éstos llevan a los requisitos, se puede encontrar en el Anexo II.

### 2.1. Motivación

La motivación del proyecto surge de la necesidad de una vía de caracterización de un jet de gas basada en interferometría, que permita la obtención de una imagen del índice de refracción del jet de gas, de una forma ágil, todo ello mientras se realizan experimentos con el jet, como aceleración de electrones.

Este tipo de experimentos está en una fase temprana, por lo que todavía no se realizan de forma exhaustiva. Es por ello que para los trabajos que se han realizado hasta ahora con el jet de gas, se han obtenido las imágenes del jet a partir de imágenes tomadas directamente con la cámara, y se han procesado una por una y manualmente, con un software de procesado de imágenes de interferencia [4].

Este proceso para el tratamiento de las imágenes no se podía describir con otro adjetivo que no fuera “tedioso”, además de que no permitía la caracterización del jet a tiempo real, característica imprescindible para poder continuar realizando experimentos con el jet de gas.

De ahí la necesidad de una software que automatizara todas esas tareas, para que el usuario final pueda simplemente, tras haber fijado unos parámetros, observar una buena imagen del jet de gas a tiempo real sobre el experimento, o bien, aplicar a posteriori el mismo procesado a series completas de imágenes.

## 2.2. *Objetivos funcionales y no funcionales*

Los objetivos funcionales y no funcionales de la aplicación creada se pueden resumir en los siguientes:

- **Obtención de una imagen del jet de gas:** la aplicación deberá ser capaz de realizar todas las operaciones necesarias para obtener una imagen nítida de un jet de gas a partir de la imagen tomada en un sistema de interferometría, junto con una imagen de referencia, así como de representarla con distintos gráficos, y poder exportar todos los resultados obtenidos.
- **Primer procesado o pre-procesado:** la aplicación deberá ser capaz de obtener una primera imagen del jet de gas, a partir de una referencia, junto con unos parámetros de cálculo y una imagen, o bien tomada desde un archivo, o bien tomada directamente desde la cámara.
- **Segundo procesado:** la aplicación deberá ser capaz de, una vez obtenida una imagen del jet, tratarla mediante distintos algoritmos para eliminar ruido, o defectos en la imagen, o simplemente aplicarle distintas operaciones que el usuario desee, para poder llegar entre otros objetivos, a una imagen del índice de refracción del jet de gas en cada punto.
- **Tiempo real:** la aplicación deberá poder realizar y mostrar los cálculos antes mencionados a la mayor velocidad posible y a tiempo real, según va tomando las imágenes de la interferencia en el sistema. Idealmente debería procesar 10 imágenes por segundo, aunque para la realización de experimentos sería admisible hasta un mínimo de una imagen por segundo. Para ello se utilizará procesamiento con CUDA de NVIDIA que permitirá realizar los cálculos necesarios de una forma mucho más rápida, aunque se mantendrá la posibilidad de cálculo con la CPU para los equipos que no dispongan de una tarjeta compatible.
- **Exportado de resultados:** el sistema incluirá la opción de exportar todos los resultados obtenidos en diferentes formatos, tanto los gráficos (imagen, vídeo,

animación), como los datos numéricos de las imágenes procesadas en texto plano.

- **Gestión de la cámara:** el sistema deberá poder configurar la cámara para la adecuada toma de imágenes a gusto del usuario.
- **Gestión de proyectos:** el sistema deberá ser capaz de tratar con proyectos completos, para que el usuario pueda continuar el proyecto por donde lo dejó el día anterior de forma rápida.
- **Usabilidad del programa:** la aplicación que se cree debe ser cómoda de utilizar y reunir todas las características de usabilidad que se suele encontrar en el software comercial, de forma que la experiencia del usuario sea lo mejor posible, permitiendo siempre varias formas de realizar la misma operación, mostrando alertas de errores o de aviso de guardado del proyecto, mostrando una barra de estado con información de la ejecución actual..., y siempre automatizando todos los procesos en la medida de lo posible.

En resumen, se busca crear una aplicación completa orientada a un usuario con el perfil de investigador, capaz de obtener y procesar adecuadamente una imagen del jet de gas, a partir de las imágenes, o bien tomadas directamente con la cámara, o bien cargadas de archivo y en la que prime la usabilidad de forma que haga el trabajo al investigador lo más fácil posible, permitiéndole generar incluso gráficos adecuados para su uso en pósters o publicaciones, además de obtener los resultados en formatos de texto que permitan su posterior uso en aplicaciones como MATLAB® o Wolfram Mathematica®.

Además, esta aplicación, aunque en un primer momento deberá funcionar bajo los sistemas Microsoft Windows, debe quedar lo más abierta posible a su uso en otras plataformas, por lo que solamente se utilizarán bibliotecas multiplataforma, que reduzcan el proceso de migración a otras plataformas a una simple compilación y enlazado adecuados, sin tener que tocar el código.

### ***2.3. Objetivos técnicos***

Llevar a cabo este proceso de construcción de una aplicación completa, va a suponer para el autor formación en los siguientes campos/herramientas:

- Estudio de la física necesaria para el procesado de las imágenes.
- Estudio de los patrones y técnicas de ingeniería para un correcto diseño del programa que permita añadir módulos o cambiar la interfaz sin que haya que tocar más que algunas partes aisladas del código.
- Programación en C++ para la generación del código de la aplicación.
- Estudio de la biblioteca de Qt para el desarrollo de la interfaz gráfica, así como el trabajo en paralelo con hilos.
- Estudio de la herramienta CUDA de NVIDIA para el procesamiento en paralelo de las imágenes con la tarjeta gráfica: de su programación y de su integración con Qt, siendo este proceso bastante delicado, ya que NVIDIA no da soporte oficial para este entorno de desarrollo.
- Estudio de las bibliotecas de representación multiplataforma QwtPlot y QwtPlot3D. Aunque este proceso pueda parecer fácil, ha requerido bastante tiempo la creación de clases que permitan la representación de diversos tipos de gráficos, teniendo en cuenta que estos gráficos tienen que soportar opciones de escalado adecuadas, así como la representación de secuencias de imágenes, evitando parpadeos en la medida de lo posible, y haciendo el proceso lo más eficiente posible.
- Aprendizaje de uso de la biblioteca de compresión de archivos ZIP, QuaZIP.
- Aprendizaje de uso de herramientas para la creación de vídeos y animaciones.

### 3. Conceptos teóricos

En los siguientes apartados se exponen los aspectos claves para la comprensión teórica del proyecto.

Se comenzará por aspectos más generales acerca de la física o las matemáticas detrás de los cálculos utilizados para el procesamiento, para continuar con aspectos ya orientados a los propios procesos y secuencias de procesos necesarios desde la toma de la imagen de interferencia hasta la obtención y filtrado de la imagen del jet de gas.

El capítulo finalizará con la descripción del experimento sobre el que está colocado el jet de gas, mostrando brevemente el montaje óptico, y algunos resultados.

Nótese que este capítulo pretende, por una parte familiarizar al lector con la temática del proyecto y por otra describir las operaciones que se realizan sobre la imagen. Se recomienda al menos una lectura rápida a los apartados 3.4 y 3.5, ya que describen paso a paso el proceso que se realiza desde que se toma una imagen con la cámara, hasta que se obtiene una buena imagen del jet de gas.

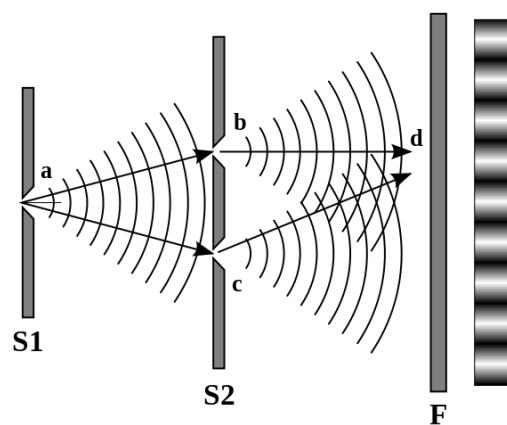
#### ***3.1. La luz como onda: Interferometría***

Desde que en 1690, el holandés Christiaan Huygens publicara su teoría sobre la naturaleza ondulatoria de la luz, existió una gran discusión acerca de si la luz consistía en partículas, como afirmaban Pierre Gassendi o Isaac Newton; o en ondas.

El porqué de esta discusión se debía a que, cuando se estudiaba la luz, se encontraban resultados que hacían pensar que la luz era una onda, como la difracción en el experimento de las franjas de Young (Figura 3), en contraposición a experimentos que hacían pensar que la luz estaba formada por partículas, como la radiación del cuerpo negro en cuantos de energía o el efecto fotoeléctrico.

Finalmente, en 1924, Louis-Victor de Broglie, planteó la dualidad onda-corpúsculo, que más tarde fue apoyada por el mismo Albert Einstein, y que trata a toda la materia, y en

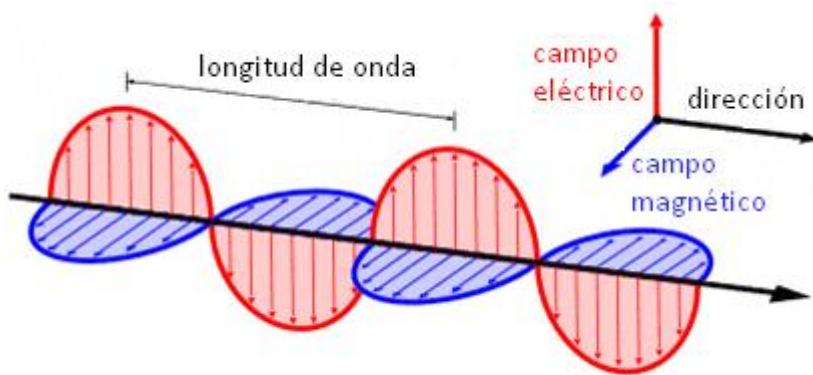
particular a la luz, como mezcla de onda y partícula, pudiendo manifestar por lo tanto comportamiento de ambos tipos.



*Figura 3. Experimento de la doble rendija de Young*

En lo que a este proyecto se refiere, estamos interesados en la naturaleza ondulatoria de la luz.

Una onda electromagnética, y en particular la luz, es la propagación de campos eléctricos y magnéticos, a través del vacío u otro medio, transportando energía.



*Figura 4. Onda electromagnética monocromática*

El ejemplo de luz más sencillo es la luz monocromática, que es aquella en la que los campos eléctricos y magnéticos varían siempre con una tasa de repetición constante llamada frecuencia, como la que se puede observar en la Figura 4.

Está demostrado que estas ondas electromagnéticas en el vacío siguen las leyes de Maxwell.

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_0}$$

$$\vec{\nabla} \cdot \vec{B} = 0$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

$$\vec{\nabla} \times \vec{B} = \mu_0 \vec{J} + \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t}$$

*Figura 5. Leyes de Maxwell en el vacío*

Que en particular, para la propagación en una zona en el que no hay fuentes, se reducen a eliminar los términos con  $\rho$  y  $\vec{J}$ :

$$\vec{\nabla} \cdot \vec{E} = 0$$

$$\vec{\nabla} \cdot \vec{B} = 0$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

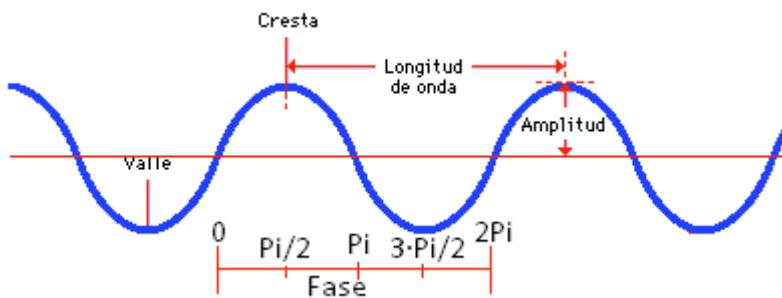
$$\vec{\nabla} \times \vec{B} = \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t}$$

*Figura 6. Leyes de Maxwell en el vacío sin fuentes*

Y de donde se puede deducir que en la propagación, los campos eléctricos y magnéticos son perpendiculares entre sí, y que por lo tanto en la mayoría de los casos bastará con el estudio de uno de ellos, sabiendo que el otro aparecerá perpendicular al primero.

Analicemos ahora una figura del campo eléctrico propagándose como el que se puede ver en la Figura 7.

Ya mencionamos antes que la frecuencia es la tasa de repetición con la que varía el campo, y por tanto se mide en Hercios (Hz) o repeticiones por segundo (1/s). Por lo que la siguiente cantidad a definir es la longitud de onda.



*Figura 7. Campo eléctrico propagándose*

Dado que la onda es algo que se propaga, la longitud de onda es la distancia que le da tiempo a la onda a propagarse durante uno de los ciclos de repetición, es decir, por ejemplo de máximo a máximo, y por lo tanto se calculará como  $\lambda = v/f$  donde  $v$  es la velocidad de propagación de la luz por el medio, y  $f$  la frecuencia. Esta velocidad de propagación en el medio se puede expresar en función de la velocidad de la luz en el vacío  $c$ , definiendo el índice de refracción como  $n = \frac{c}{v}$ . Nótese que la longitud de onda se hará más corta cuando la velocidad de propagación en el medio sea más baja, y que esta velocidad de propagación va inversamente relacionada con la densidad del medio.

Por otra parte la fase indica en qué parte del ciclo se encuentra la onda en cada instante y en cada punto del espacio. Se suele medir en radianes, tomando valores entre 0 y  $2\pi$ , coincidiendo el 0 con el principio del ciclo, y el  $2\pi$ , con el final del ciclo, y por lo tanto el 0 del siguiente ciclo.

Este será el contexto para entender el fenómeno de la interferencia.

Supongamos que tenemos dos ondas de luz de la misma frecuencia y amplitud, pasando por la misma parte del espacio y tengamos en cuenta que los campos eléctricos y magnéticos cumplen el principio de superposición, es decir, que se pueden sumar en cada punto del espacio.

Se dice que las dos ondas están en fase cuando ambas van “sincronizadas”, sin diferencia de fase entre ellas (o diferencias de  $2\pi k$ ), como se muestra en la parte derecha de la Figura 8, en cuyo caso, la onda final es una onda de amplitud doble e intensidad máxima: interferencia constructiva.

Por otra parte, en la izquierda, observamos dos ondas en contrafase, con diferencia de fase de  $\pi$  (o  $\pi + 2\pi k$ ), por lo que al sumarlas, el resultado es amplitud cero e intensidad nula: interferencia destructiva.

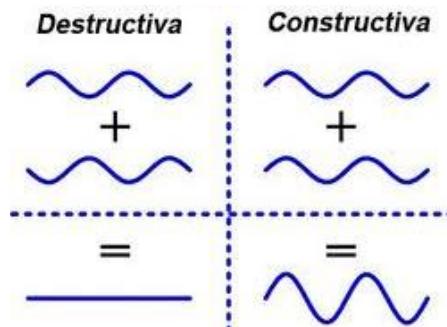


Figura 8. Interferencia constructiva y destructiva

Y además están todos los casos intermedios de diferencias de fases intermedias, que proporcionarán valores intermedios para la intensidad.

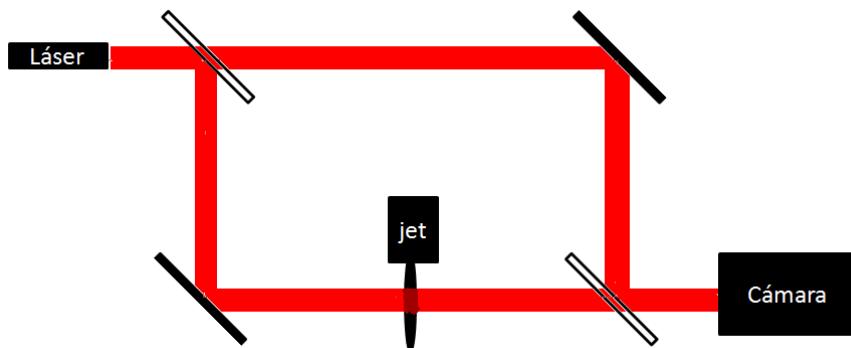
### 3.1.1. Interferómetro de Mach-Zehnder

La idea de la interferometría usada es que ahora, en vez de tener dos ondas de luz, se tiene todo un frente plano de ondas que se divide en dos ramas, haciendo pasar solamente una de ellas por el medio o material cuya densidad se quiere conocer, mientras que la otra continua por el vacío o el medio del ambiente.

De esta forma el frente que pasa por el material, con densidad y por tanto índice de refracción distinto, se frenará ligeramente y se retrasará respecto al otro, perdiendo la sincronización de fase.

Si a continuación se vuelve a juntar cada uno de los frentes en uno solo, y se mide la intensidad del frente resultante en cada punto con una cámara, se podrá recuperar la información de la diferencia de fase en cada punto, y así saber cuánto se ha frenado el

frente que ha pasado por el medio que se quiere medir, pudiendo recuperar así el índice de refracción de dicho medio.



*Figura 9. Interferencia de dos frentes de onda*

Existen diferentes tipos de interferómetros con distintas disposiciones que utilizan esta idea. El más indicado para este caso, en el que se busca medir pequeños cambios de fase del orden de  $\pi$ , es el interferómetro de Mach-Zehnder.

El interferómetro de Mach-Zehnder, utiliza la misma idea que la Figura 9 pero inclinando ligeramente una de las dos ramas. Eso provoca que aparezcan unas franjas en el patrón de interferencia, debido a que en esta situación la interferencia no aparece sólo debido al paso por el medio, sino a la propia inclinación de una de las dos ramas.

La desventaja ahora es que al medir la fase final, se encontrarán por tanto cambios de fase debidos tanto al paso por el medio, como a la inclinación introducida. El cambio de fase debido a la inclinación no es de interés en los resultados y para eliminarlo es necesario tomar una imagen de referencia de la interferencia cuando ninguna de las dos ramas pasa por el medio, calcular su fase, y restársela a la fase obtenida cuando una de las dos ramas sí pasa por el medio. No obstante, este proceso no es sólo una desventaja, ya que también ayuda a eliminar errores instrumentales.

De esta forma, se obtendrá finalmente el cambio de fase al pasar por el medio, proceso que se detallará a efectos prácticos en el capítulo 3.4.

Fuentes:[5][6][7][8]

### **3.2. Transformada de Fourier Bidimensional Compleja**

#### **3.2.1. Transformada de Fourier Unidimensional**

La transformada de Fourier unidimensional es una transformación que hace corresponder a una función compleja otra función compleja mediante la siguiente fórmula:

$$F(\xi) = \int_{-\infty}^{\infty} f(x) \cdot e^{-2\pi x \xi i} dx$$

*Figura 10. Fórmula de cálculo de la transformada de Fourier*

Tiene la propiedad de ser biunívoca, y que por lo tanto, a partir de la función transformada se puede obtener de nuevo la función original mediante la transformada inversa:

$$f(x) = \int_{-\infty}^{\infty} F(\xi) \cdot e^{2\pi x \xi i} d\xi$$

*Figura 11. Fórmula de cálculo de la transformada Fourier inversa*

La variable de la que depende la función transformada se denomina variable conjugada, y en general, en física, esta transformación tiene interés cuando esa variable conjugada representa alguna magnitud física.

Uno de los ejemplos más comunes son el tiempo y la frecuencia. De toda función integrable dependiente del tiempo, se puede calcular su transformada de Fourier, para en su lugar, obtener otra función con dependencia en frecuencia, que contiene exactamente la misma información, pero organizada de otra forma, y que puede ser útil para por ejemplo, el filtrado en frecuencia, como el que hace por ejemplo la radio, cuando sintoniza un canal concreto, evitando que suenen interferencias procedentes de otras emisoras.

Por último cabe destacar que la transforma de Fourier de una función seno o coseno, sólo contiene componentes en exactamente los valores de la frecuencia del seno o

coseno como se puede ver en las siguientes figuras. Esto hace pensar en cualquier función como una composición de funciones seno y coseno de un número infinito no numerable de frecuencias, con distintas amplitudes y fases entre sí.

Así, una vez calculada la transformada de Fourier, cada frecuencia tendrá asociada un número complejo, cuyo módulo representará la amplitud y cuyo argumento representará la fase de una función seno asociada a esa misma frecuencia.

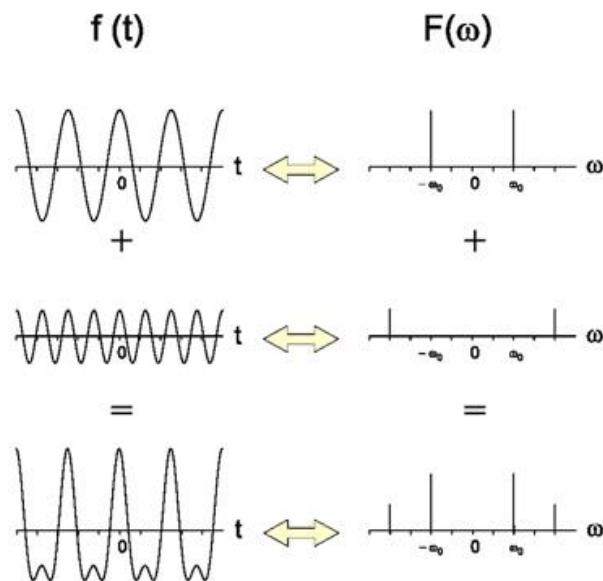


Figura 12. Ejemplos de módulos de funciones y sus transformadas

### 3.2.2. Transformada de Fourier Espacial Bidimensional

De forma análoga, se puede definir la transformada de Fourier espacial bidimensional como la que asocia a una función compleja dependiente de dos variables espaciales, otra función compleja dependiente de dos variables conjugadas, llamadas frecuencias espaciales:

$$F(u, v) = \iint_{-\infty}^{\infty} f(x, y) \cdot e^{-2\pi(ux+vy)i} dx dy$$

Figura 13. Ecuación de la transformada de Fourier bidimensional

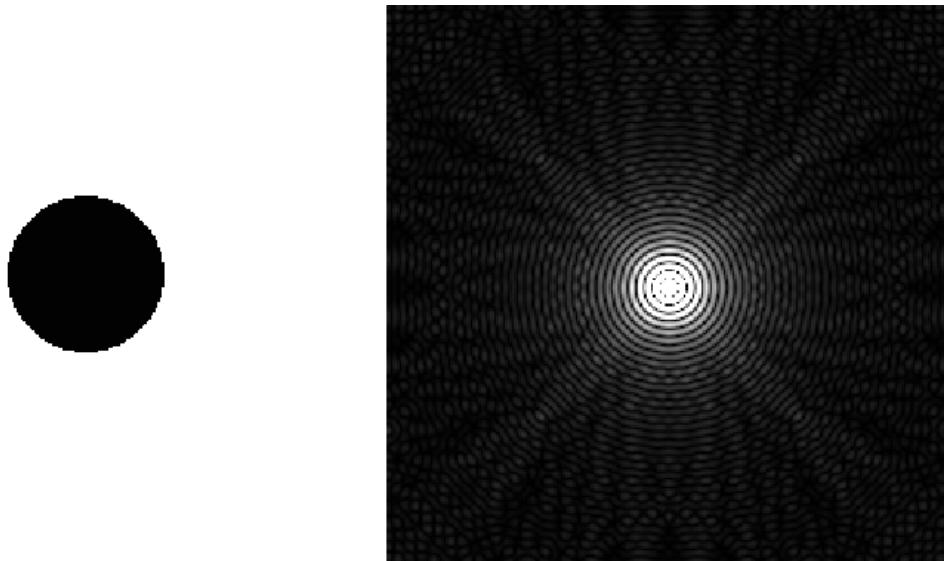
Y su inversa:

$$f(x, y) = \iint_{-\infty}^{\infty} F(u, v) \cdot e^{2\pi(ux+vy)i} du dv$$

*Figura 14. Ecuación de la transformada de Fourier bidimensional inversa*

Un caso particular de función de dos variables es la intensidad en cada punto de una imagen, supuesta continua y en escala de grises (Dado que una imagen en color son realmente tres imágenes, una con cada componente de color).

Esto hace que se pueda calcular la transformada de Fourier de una imagen:



*Figura 15. Ejemplo de transformada de Fourier de una imagen*

Donde la variable conjugada al eje “x”, será la frecuencia espacial horizontal “u” y de la misma forma la variable conjugada a la distancia “y”, será la frecuencia espacial vertical “v”, y representarán, de la misma forma que en el caso unidimensional, la frecuencia de repetición de patrones de funciones seno y coseno de distintas frecuencias, solo que ahora, la variable “u” representará los patrones en dirección horizontal, y la variable “v”, los patrones en dirección vertical.

### 3.2.3. Transformada de Fourier Rápida (FFT)

Hasta ahora hemos tratado con funciones continuas, sin embargo, en informática, siempre se trata con sistemas discretos (o continuos muestreados), y en el caso de una

imagen, sólo se toman valores de la intensidad para un cierto conjunto de puntos denominados píxeles.

La distancia entre los píxeles, por ejemplo en el eje x, vendrá dada por:

$$dx = \frac{x_{max} - x_{min}}{N - 1}$$

Siendo N el número de muestras tomadas. Esto además implicará que la mayor frecuencia que puede contener la imagen, es la que hace variar la intensidad abruptamente de un punto a otro, es decir, con una periodicidad del doble de la distancia entre píxeles:

$$u_{max} = \frac{1}{2 \cdot dx}$$

En ese caso, se utiliza la transformada de Fourier discreta, cuyas fórmulas vienen dadas por:

$$F_k = \sum_{n=0}^{N-1} f_n \cdot e^{-j2\pi(\frac{kn}{N})}$$

*Figura 16. Ecuación de la transformada de Fourier unidimensional discreta*

$$F_{kl} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_{nm} \cdot e^{-j2\pi(\frac{kn}{N} + \frac{lm}{M})}$$

*Figura 17. Ecuación de la transformada de Fourier bidimensional discreta*

No obstante estas ecuaciones son muy lentas de evaluar para todos los puntos, en particular un orden  $n^2$  para la primera ecuación y  $n^4$  para la segunda. Sin embargo, existe un algoritmo llamado “Transformada Rápida de Fourier” o FFT (Fast Fourier Transform) que permite realizar el cálculo de una forma muy eficiente, mediante una serie de superposiciones de la imagen sobre sí misma, reduciendo el orden a  $n \cdot \log(n)$  y  $n^2 \cdot \log(n)$  respectivamente.

Este algoritmo asume que la imagen se repite periódicamente en todas las direcciones y que por lo tanto su transformada de Fourier, como para toda función periódica, también es discreta, es decir, que sólo toma valores para determinadas frecuencias  $f = \frac{n}{T}$  siendo  $n$  un número natural y  $T$  la periodicidad de la función en la dirección de la variable en que nos estamos fijando.

Además la imagen de la transformada mantiene la resolución de la original y también existen frecuencias negativas, por lo que la frecuencia máxima que se podrá representar será  $f = \frac{N}{2T}$ , siendo  $N$  el número de muestras en la dirección que corresponda.

Por lo tanto, la imagen final, con la misma resolución que la inicial tendrá unas diferencias de frecuencia entre pixel y pixel en la dirección horizontal de:

$$du = \frac{1}{x_{max} - x_{min}} = \frac{1}{dx \cdot (N - 1)}$$

Y entonces la frecuencia máxima que estaremos representando será la mitad (pues hay tantas frecuencias negativas como positivas) del producto entre el número de píxeles menos 1 y la distancia entre cada pixel:

$$u_{max} = \frac{du \cdot (N - 1)}{2} = \frac{1}{2 \cdot dx}$$

Que es precisamente la máxima frecuencia que podía contener nuestra imagen, por lo que en principio el algoritmo se podría ejecutar sin pérdida de información.

Por otra parte, la eficiencia del algoritmo viene condicionada con una serie de restricciones de entrada y de salida:

- La imagen de entrada debe de tener un número de píxeles que sea potencia de 2 tanto en la dirección horizontal como en la vertical, es decir, 1, 2, 4, 8, 16..., esto hace posible la estrategia de superposición con la propia imagen.

- La imagen de salida con la transformada, está desordenada y hay que ordenarla y volverla a desordenar antes de realizar la transformada inversa. Esta ordenación se puede apreciar en la Figura 18.

Sin embargo, estos dos detalles, aunque algo molestos, no impiden que el algoritmo FFT suponga una mejora increíble al cálculo directamente con la fórmula, ya que el primero de ellos solo implica el uso de una mayor cantidad de memoria, y el segundo, aplicar una operación de recolocado de orden menor al del propio algoritmo.

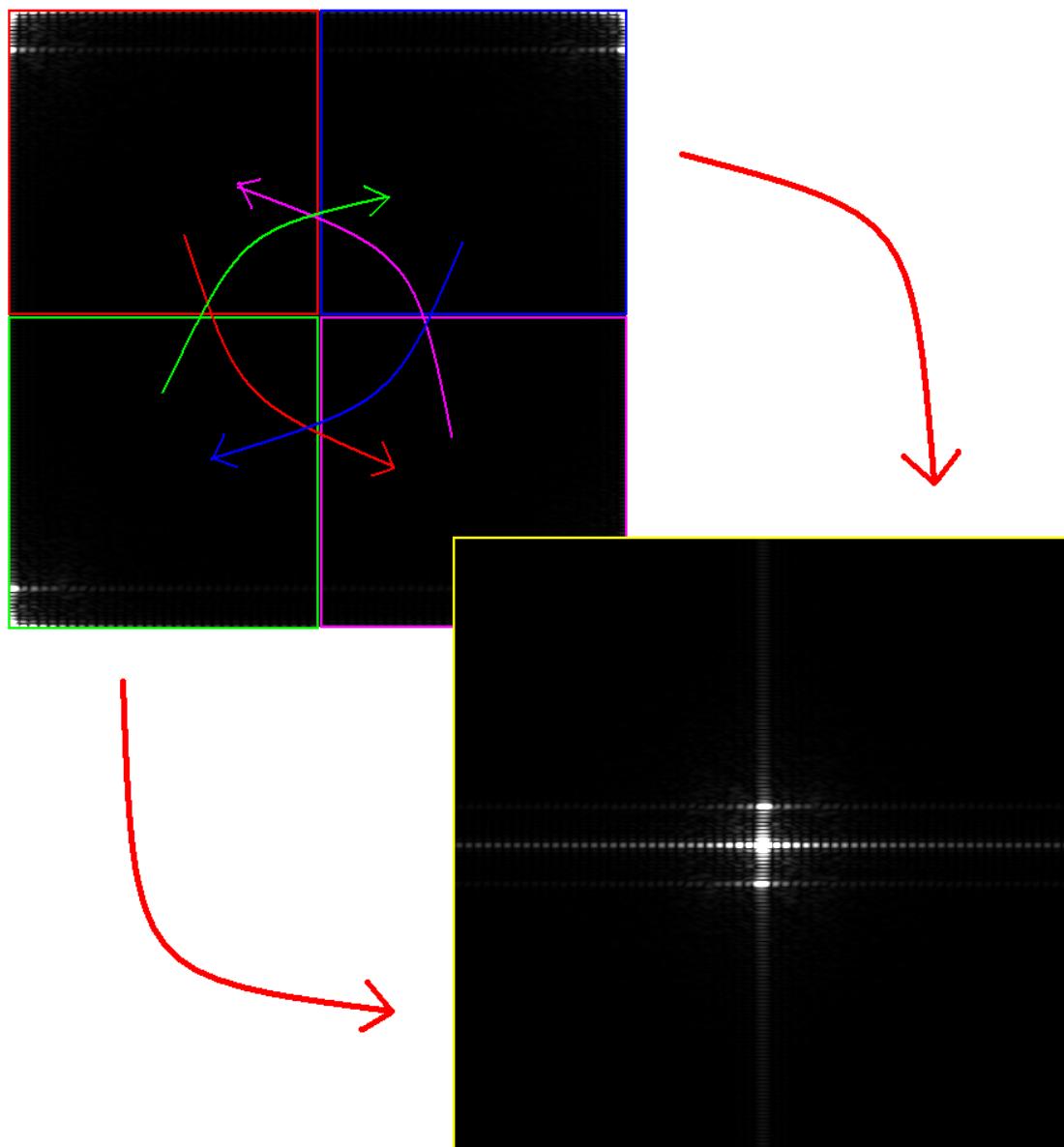


Figura 18. Reordenación de la transformada de Fourier tras el algoritmo FFT

Para detalles concretos de la implementación de la transformada de Fourier con el algoritmo FFT consúltese [9].

Fuentes:[7] [9] [10]

### 3.3. Regresión lineal: Ajuste a mínimos cuadrados

Una de las operaciones que se realizarán para mejorar la imagen obtenida, es eliminar la inclinación lineal de la imagen. Este capítulo trata acerca de cómo obtener, la inclinación lineal de la imagen, para poder eliminarla.

El objetivo por ahora es encontrar la inclinación media que tienen los puntos a lo largo de una recta. Supongamos que tenemos una serie de puntos distribuidos como en la Figura 19 y queremos calcular la pendiente media.

Para ello, lo más habitual es ajustar los puntos a una recta de la forma  $y = m \cdot x + n$  y ver cuál es la pendiente de dicha recta.

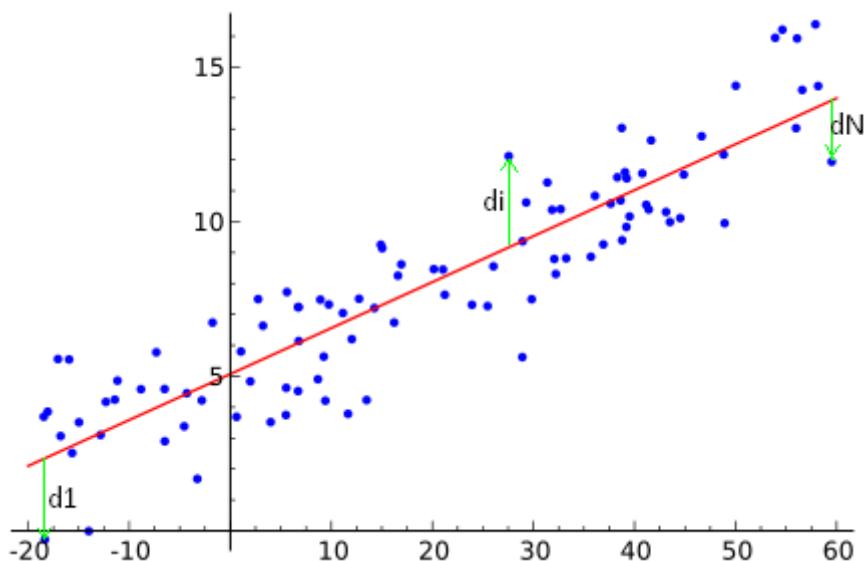


Figura 19. Regresión lineal

El ajuste a esa recta, se realiza por el método de los mínimos cuadrados. Este método consiste en buscar la recta de forma que la suma de los cuadrados de las distancias desde cada punto a la recta sea mínima, es decir, la recta que minimiza esta cantidad:

$$D_N^2 = \sum_{i=1}^N d_i^2 = \sum_{i=1}^N (y_i - (m \cdot x_i + n))^2$$

Donde  $x_i, y_i$  son las coordenadas de cada uno de los puntos a los que queremos ajustar la recta.

Para ello, derivando por una parte respecto a  $n$  e igualando a 0, y de la misma forma para  $m$ , se llega a un sistema de ecuaciones para  $m$  y  $n$  cuya solución es:

$$m = \frac{\sum x_i \cdot \sum y_i - N \cdot \sum (x_i \cdot y_i)}{(\sum x_i)^2 - N \cdot \sum y_i^2}$$

$$n = \frac{\sum y_i - m \cdot \sum x_i}{N}$$

*Figura 20. Ecuaciones de la regresión lineal*

Que son los parámetros de la recta a la que mejor se ajustan los puntos y si, como en este caso sólo interesa calcular la inclinación, basta con la expresión de arriba.

Fuentes: [11]

### 3.4. Pre-procesado de una imagen con una referencia

En el capítulo 3.1 se explicó en qué consiste la idea de la interferometría, y en qué consiste un interferómetro de Mach-Zehnder.

Este apartado es una guía paso a paso de todas las operaciones que hay que realizar para pasar desde una imagen de la interferencia tomada con la cámara en el sistema, hasta la obtención de una imagen del jet: lo que denominamos pre-procesado.

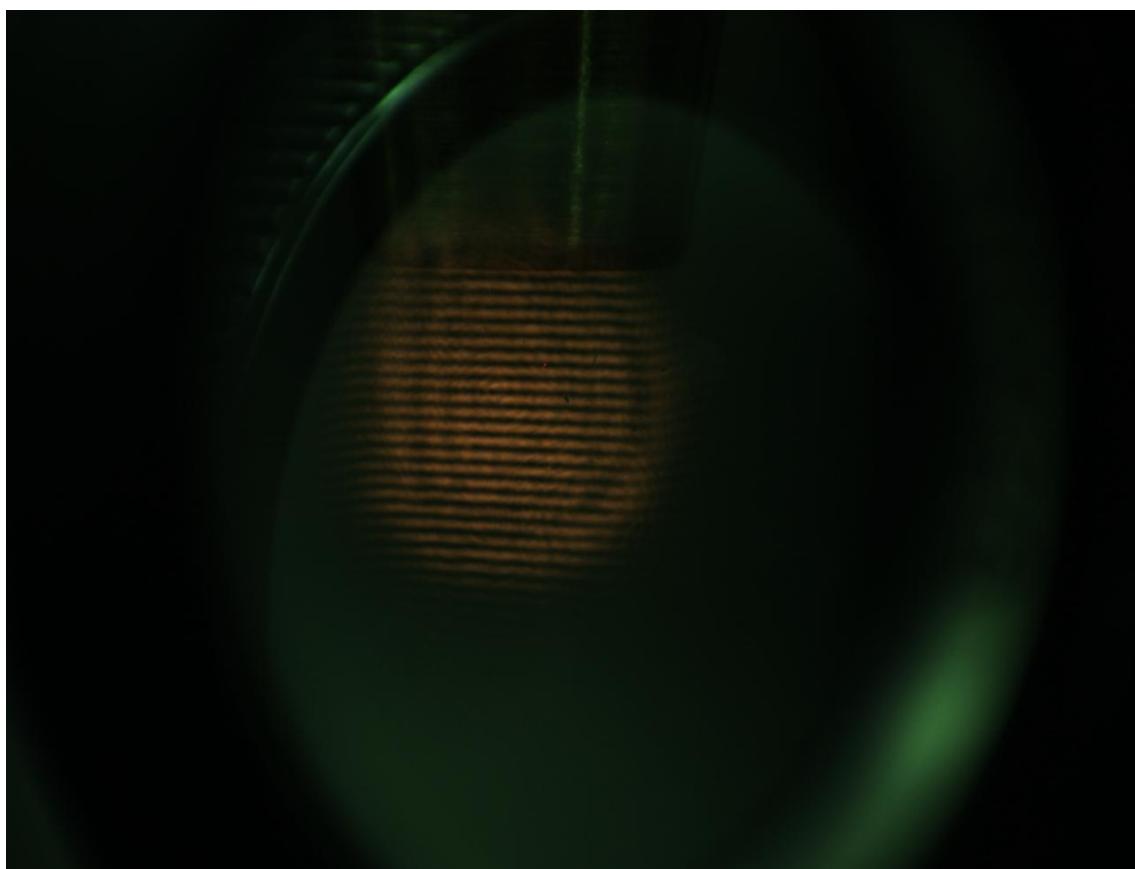
Como ya se comentó, en un interferómetro de ese tipo, es necesario además de la imagen de la interferencia con el jet de gas encendido, otra imagen de referencia en ausencia del jet de gas.

### 3.4.1. Obtención de la fase

Tal y como se explicó en el capítulo 3.1, es necesario conocer la fase que lleva la rama del láser que atraviesa el jet de gas, para averiguar el índice de refracción del jet de gas, y consecuentemente la densidad.

Este primer proceso de cálculo de la fase tiene que realizarse tanto para la referencia como para la imagen del jet.

En primer lugar se toma la imagen con las franjas de interferencia adquirida por la cámara, como la que se puede ver en la Figura 21, imagen de 1280x1024 píxeles.



*Figura 21. Imagen de la interferencia adquirida por la cámara*

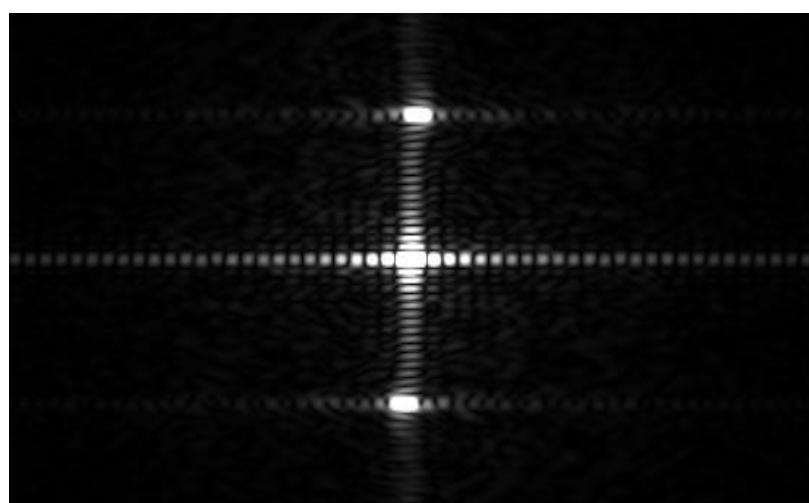
A continuación, se convierte la imagen a escala de grises y se redimensiona para que tenga unas dimensiones que potencia de dos, en este caso, por ejemplo, redondeando por arriba los valores de la imagen original, obteniendo una imagen de 2048x1024 píxeles.

Además se aplica una máscara para establecer a negro todo menos la zona de las franjas como se puede observar en la Figura 22. Nótese que cuanto mayor sea el tamaño de la imagen total, aunque contenga extensas zonas negras, mayor resolución tendrá la transformada de Fourier.



*Figura 22. Imagen ya adecuada para el pre-procesado*

El siguiente paso en el proceso consiste en calcular la transformada de Fourier de la imagen considerando que dicha imagen es la parte real de la función inicial, siendo la parte imaginaria 0. Tras realizar esto, y representando el valor absoluto para la transformada se obtiene el resultado de la Figura 23.



*Figura 23. Transformada de Fourier de la imagen*



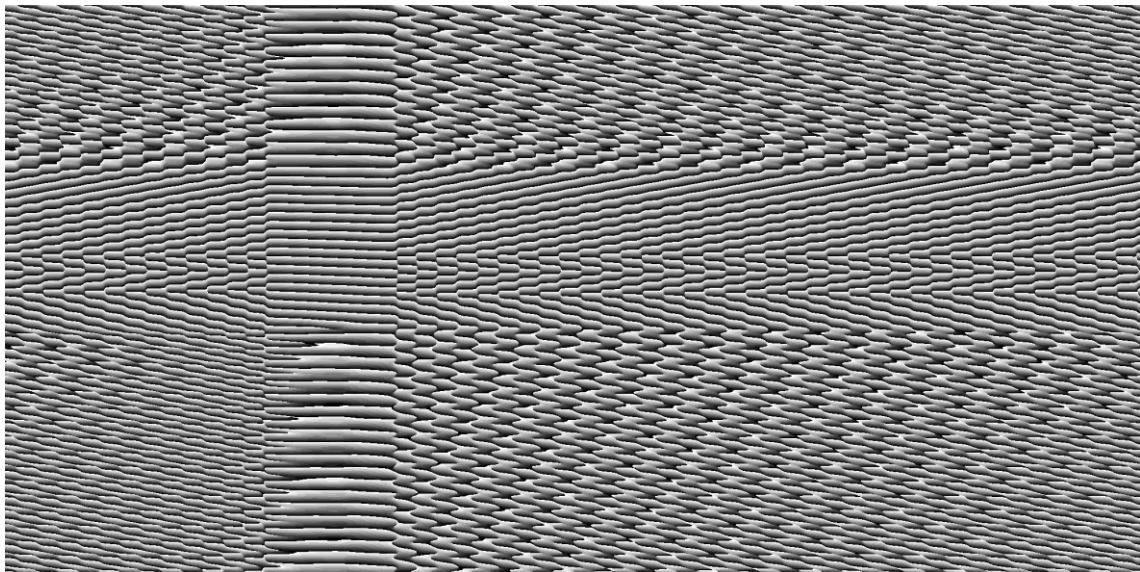
*Figura 24. Transformada de Fourier con la máscara en el término de la fase*

A continuación hay que aplicar una máscara a la transformada de Fourier para mantener sólo el máximo de arriba o el de abajo, ya que son los términos que contienen la fase, llegando a lo que se puede observar en la Figura 24.

Seguidamente se realiza la transformada de Fourier inversa de la imagen. Dado que se ha aplicado una máscara, al invertir la transformada de Fourier, se obtendrá una imagen con parte real y parte imaginaria: la fase en cada punto será el argumento del número complejo  $a + i \cdot b$ , asociado a cada pixel, entre 0 y  $2\pi$ , calculado como:

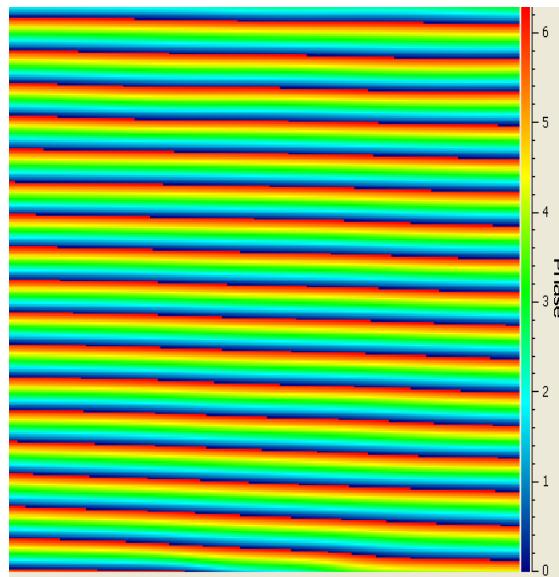
$$\varphi = \begin{cases} \operatorname{arctg}\left(\frac{b}{a}\right) & b \geq 0, a > 0 \\ \operatorname{arctg}\left(\frac{b}{a}\right) + 2\pi & b < 0, a > 0 \\ \operatorname{arctg}\left(\frac{b}{a}\right) + \pi & a < 0 \\ \frac{\pi}{2} & b > 0, a = 0 \\ \frac{3\pi}{2} & b < 0, a = 0 \\ \text{indet} & a = 0, b = 0 \end{cases}$$

De esa forma se obtiene una imagen de la fase para toda la imagen inicial entre 0 y  $2\pi$ , como la que se observa en la Figura 25.



*Figura 25. Fase tras realizar la transformada inversa*

Así que si nos quedamos de nuevo solamente con la zona en la que estábamos interesados desde un principio, y mostrándola en forma de espectrograma, se obtiene un resultado como el que se puede observar en la Figura 26.



*Figura 26. Fase en la zona de interés*

### 3.4.2. Haciendo la fase continua

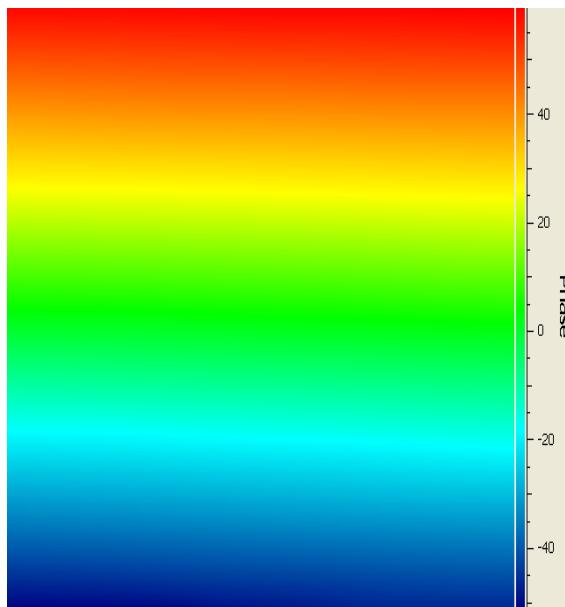
En el apartado anterior se ha llegado a obtener la fase en cada uno de los puntos de la zona de interés. Sin embargo esa fase varía sólo entre los valores 0 y  $2\pi$ . Si recordamos

en el capítulo 3.1, vimos que la fase representa el número de ciclos que se ha retrasado una de las dos ramas del láser respecto a la otra, pero que sin embargo, solo podíamos ver retrasos relativos a  $2\pi$ , porque un retraso de  $\varphi \text{ rad}$ , provoca la misma interferencia que un retraso de  $\varphi + 2\pi \text{ rad}$ , lo que hace que la fase que vemos, no sea realmente la fase de retraso.

Sin embargo, como obtuvimos las franjas debido a que inclinamos a propósito ligeramente una de las ramas, sabemos que verticalmente la fase se está acumulando linealmente, es decir, que si una franja como las que se podían ver en la Figura 26, varía entre 0 y  $2\pi$  gradualmente hacia arriba, la franja que se encuentra por encima de ella, realmente debería variar entre  $2\pi$  y  $4\pi$ , y la que se encuentra por debajo de ella entre  $-2\pi$  y 0.

Para corregir la fase utilizando esa idea, se realizará un proceso de escaneado:

- Tomaremos un punto inicial, como puede ser el punto central de la imagen y consideraremos que en esa zona la fase varía entre 0 y  $2\pi$ . Desde ese punto iremos avanzando hacia los de alrededor.
- Cada vez que encontremos un salto en la fase menor que cierto valor, como puede ser  $-3\pi/2$ , consideraremos que toda la zona más allá del salto, varía entre un intervalo  $2\pi$  mayor que la zona anterior, es decir, que si la zona anterior variaba entre 0 y  $2\pi$ , tendrá que variar entre  $2\pi$  y  $4\pi$ , por lo que a todos los puntos de la zona les sumaremos  $2\pi$ ; si en la zona anterior variaba entre  $2\pi$  y  $4\pi$ , tendrá que variar entre  $4\pi$  y  $6\pi$ , por lo que a todos los puntos de la zona les sumaremos  $4\pi$ , etc.
- De la misma forma dada vez que encontremos un salto en la fase mayor que cierto valor, como puede ser  $3\pi/2$ , consideraremos que toda la zona más allá del salto, varía entre un intervalo  $2\pi$  menor que la zona anterior, es decir, si antes variaba entre 0 y  $2\pi$ , tendrá que variar entre  $-2\pi$  y 0, por lo que a todos los puntos de la zona les restaremos  $2\pi$ ; si en la zona anterior variaba entre 0 y  $-2\pi$ , tendrá que variar entre  $-2\pi$  y  $-4\pi$ , por lo que restaremos  $4\pi$ , etc.



*Figura 27. Imagen de la fase continua tras el escaneado*

Tras acabar de procesar toda la imagen, se obtiene una imagen continua de la fase, que variará entre  $-2\pi m$  y  $2\pi n$ , y que representará el cambio de fase real.

### 3.4.3. Restando la referencia

Como ya se explicó en el capítulo 3.1.1, al inclinar una de las ramas, se introduce un gran cambio de fase debido a la inclinación, que debemos eliminar.

La forma de eliminarlo consiste en tomar dos imágenes de interferencia, una con el jet de gas en el sistema y otra sin el jet de gas, que será la imagen de referencia, y procesar las dos de la forma indicada en los dos apartados anteriores.

La imagen de la fase de referencia, sólo contendrá la fase debida a la inclinación de la rama, mientras la otra imagen de la fase contendrá la fase debida a la inclinación y además, la fase debida al jet de gas. Estas dos imágenes serán muy parecidas a la Figura 27 y casi iguales a simple vista, pero lo suficientemente diferentes entre sí debido a la presencia del jet de gas en una de ellas.

Por lo tanto, para eliminar el cambio de fase debido a la inclinación, simplemente se debe restar a la imagen de la fase con el jet de gas, la imagen de la fase de referencia. De esta forma se obtiene una imagen como la que se puede ver en la Figura 28.

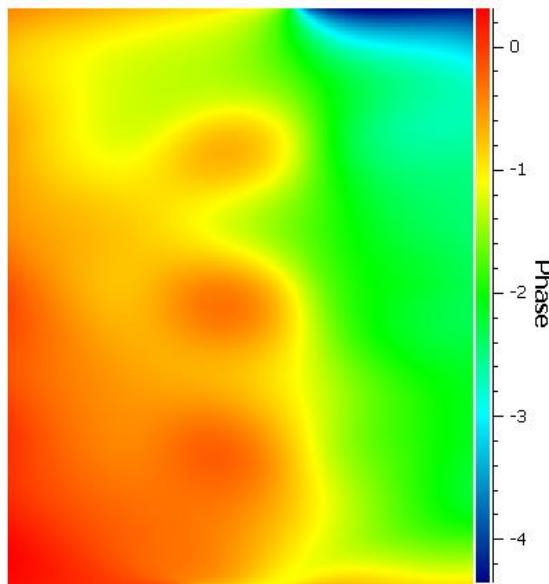


Figura 28. Primera imagen del jet de gas

Como se puede observar, ya se puede apreciar el jet de gas, sin embargo, la imagen no es buena, ya que acumula defectos, debido a pequeños movimientos en el experimento entre la toma de la imagen y de la referencia, y ruido en las medidas.

Cómo solucionar eso, e incluso aplicar otras operaciones en la imagen, se detalla en el capítulo siguiente.

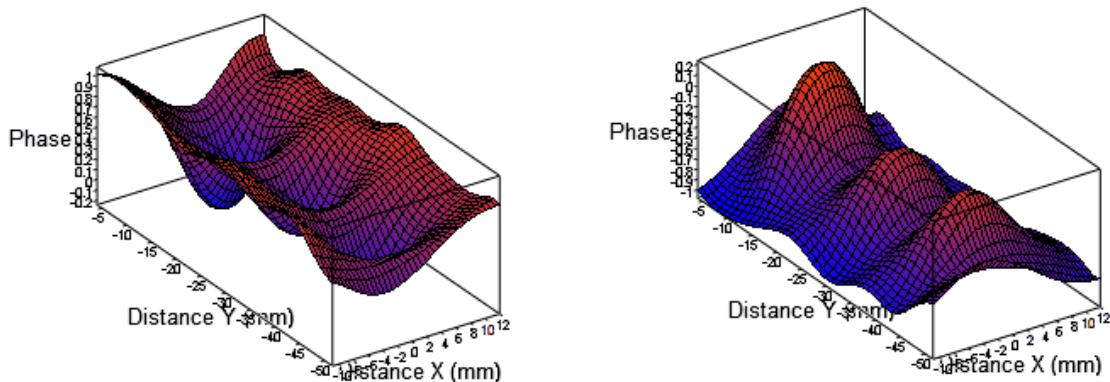
Fuentes: [8].

### **3.5. Tratamiento posterior de la imagen de la fase**

Una vez se ha obtenido una primera imagen del jet de gas, se le aplican distintas operaciones, o bien para limpiar y mejorar la imagen, o bien para obtener distintos resultados. En este capítulo se describen brevemente estas operaciones sin entrar en detalle en cómo se han implementado los algoritmos.

En la aplicación, a estas operaciones se las denominará genéricamente algoritmos, puesto que se aplican sobre la imagen de una determinada forma, aunque rigurosamente no todas son algoritmos.

### 3.5.1. Inversión de la imagen



*Figura 29. Operación de inversión, antes y después*

Esta operación invierte la imagen, es decir, transforma los valores positivos en negativos y viceversa. La importancia de esta operación radica en que en ocasiones, dependiendo del tipo de montaje, y en función de si se toma una máscara u otra en la transformada de Fourier, puede ser necesario invertir la imagen.

El resultado de la aplicación se puede observar en la Figura 29.

### 3.5.2. Eliminación de la inclinación horizontal

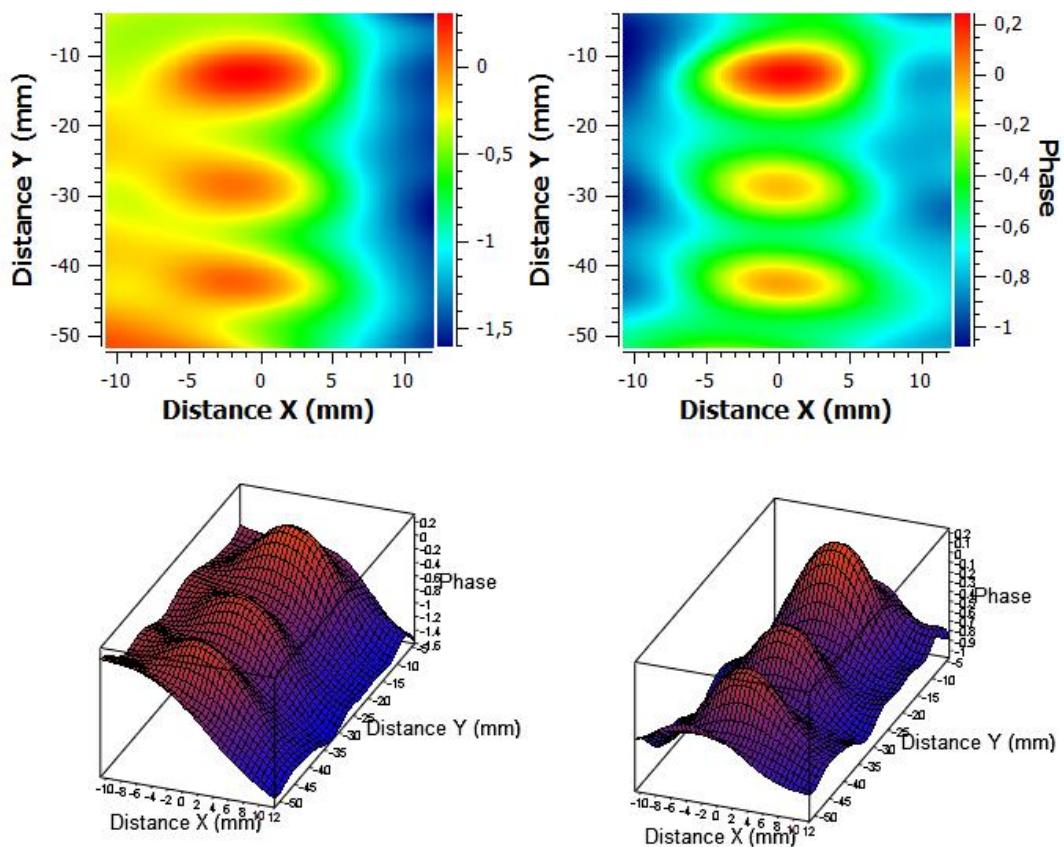
Esta operación elimina la inclinación horizontal.

En primer lugar, se calcula la pendiente para cada recta horizontal mediante un ajuste de mínimos cuadrados (Véase 3.3).

A continuación hace la media de todas esas pendientes, y se considera que esa pendiente es la inclinación lineal horizontal que posee la imagen que se pasa a eliminar aplicando a cada punto la siguiente fórmula:

$$z(x, y) = z(x, y) - \bar{m} \cdot (x - \frac{(x_{max} - x_{min})}{2})$$

El resultado de la aplicación se puede observar en la Figura 30.

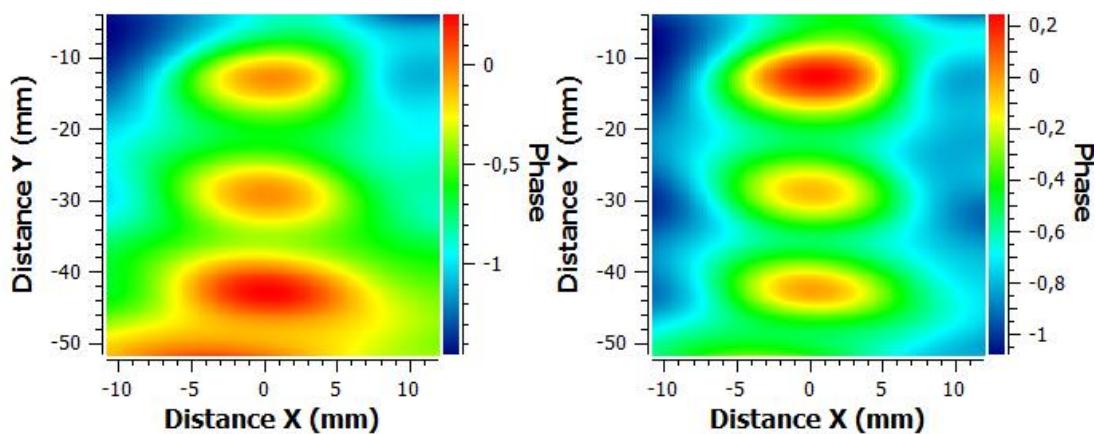


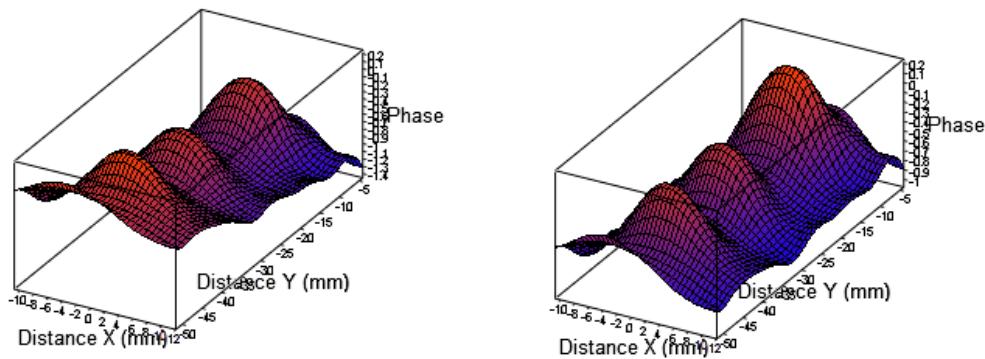
*Figura 30. Eliminación de la inclinación horizontal, antes y después*

### 3.5.3. Eliminación de la inclinación vertical

Esta operación, elimina la inclinación vertical, de la misma forma que en el apartado anterior se hacía para la inclinación horizontal.

El resultado de la aplicación se puede observar en la Figura 31.





*Figura 31. Eliminación de la inclinación vertical, antes y después*

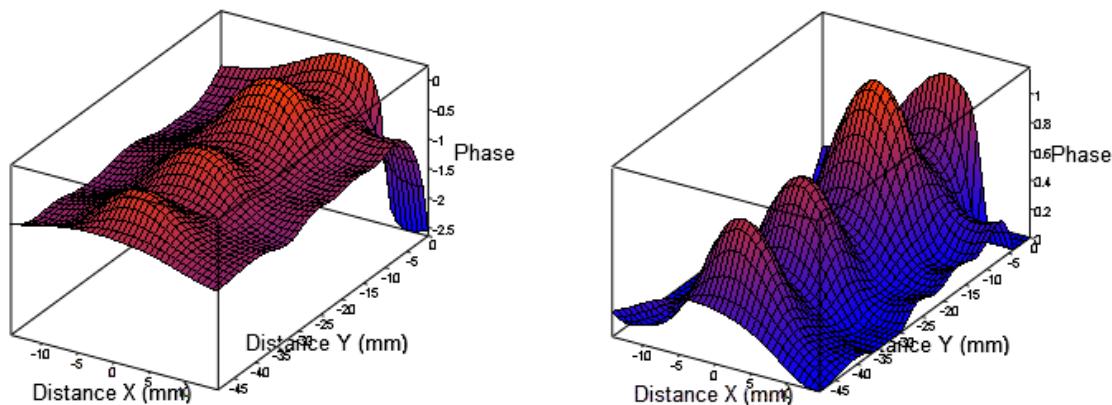
### 3.5.4. Fijado del nivel base

Esta operación elimina ruido de fondo de la imagen. Para ello, establece un valor base, promediando los valores de los puntos a lo largo primera y de la última recta vertical de la imagen, zona donde se supone que no hay jet de gas.

Una vez fijado ese valor base,  $b$ , establece ese valor, y todo lo que esté por debajo de él, como nivel base 0, mediante la siguiente fórmula:

$$z(x, y) = \begin{cases} 0, & \text{si } z(x, y) < b \\ z(x, y) - b, & \text{si } z(x, y) \geq b \end{cases}$$

El resultado de la aplicación se puede observar en la Figura 32.



*Figura 32. Operación de fijado de nivel base, antes y después*

### 3.5.5. Añadido de un valor

Esta operación simplemente suma a cada punto de la imagen un valor indicado.

El resultado de la aplicación se puede observar en la Figura 33.

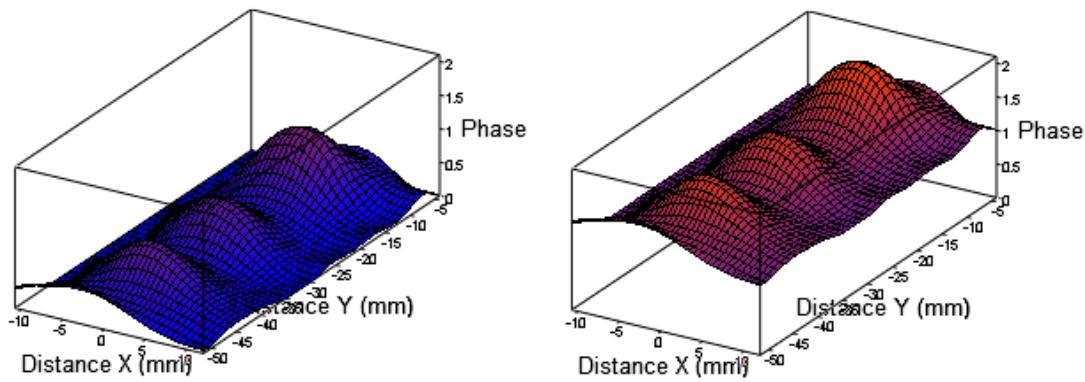


Figura 33. Operación de añadir un valor, antes y después

### 3.5.6. Multiplicación por un factor

Esta operación simplemente multiplica cada punto de la imagen por un factor indicado.

El resultado de la aplicación se puede observar en la Figura 34.

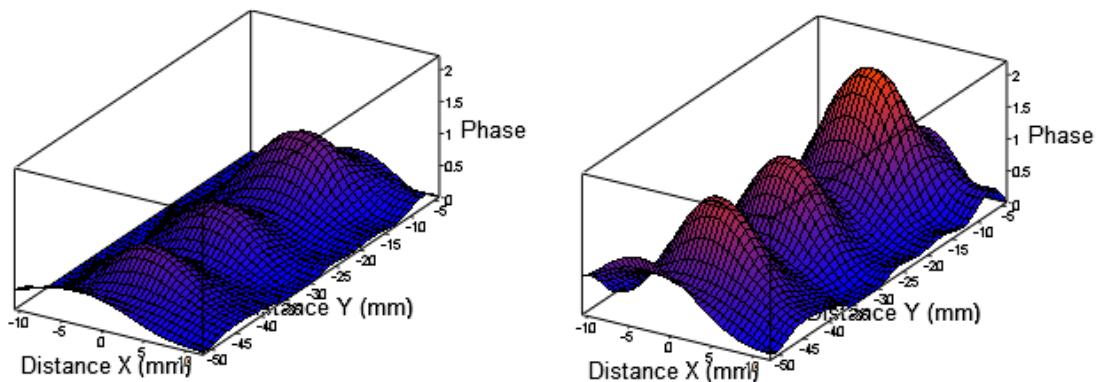


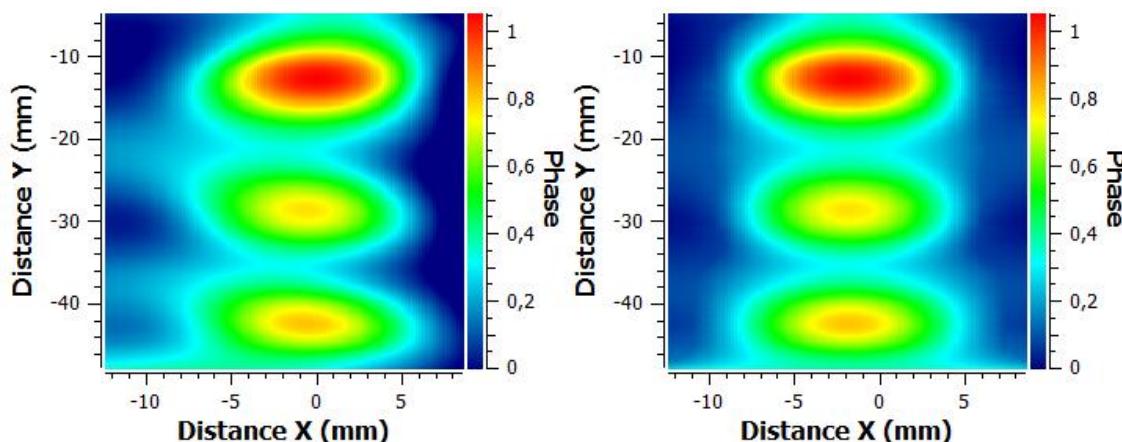
Figura 34. Operación de multiplicar por factor, antes y después

### 3.5.7. Simetrización

Esta operación, se utilizar para convertir una imagen en simétrica respecto de un eje vertical central.

Se simetriza cada recta horizontal haciendo la media con los valores a cada lado respecto del máximo y se crea una nueva imagen en la que se han colocado todas las rectas simetrizadas con el máximo situado en el centro.

El resultado de la aplicación se puede observar en la Figura 35.



*Figura 35. Operación de simetrización, antes y después*

### 3.5.8. Inversión de Abel

La inversión de Abel es una operación necesaria para conseguir recuperar la información del índice de refracción en el gas, clave para conseguir la densidad, a partir del cambio en la fase.

Cuando el láser atraviesa una sección transversal del jet de gas, suponiendo que éste presenta una simetría cilíndrica, está ocurriendo lo que se puede observar en la Figura 36.

El cambio de fase se mide, despreciando efectos de curvatura de las ondas por refracción, es realmente la proyección del cambio de fase que se ha ido acumulando en el plano de la estructura cilíndrica del jet de gas.

En la parte de la izquierda se puede observar la distribución del índice de refracción del gas, en función de la distancia al eje, y en la figura de la derecha, el cambio de fase, que se ha acumulado más en la zona central, no solamente por ser más denso en la zona central, sino también porque los rayos que atraviesan el jet de gas por el centro, lo hacen durante más longitud, de hecho, incluso aunque la distribución de índice en el gas fuera uniforme, debida a su forma cilíndrica se seguiría obteniendo una curva.

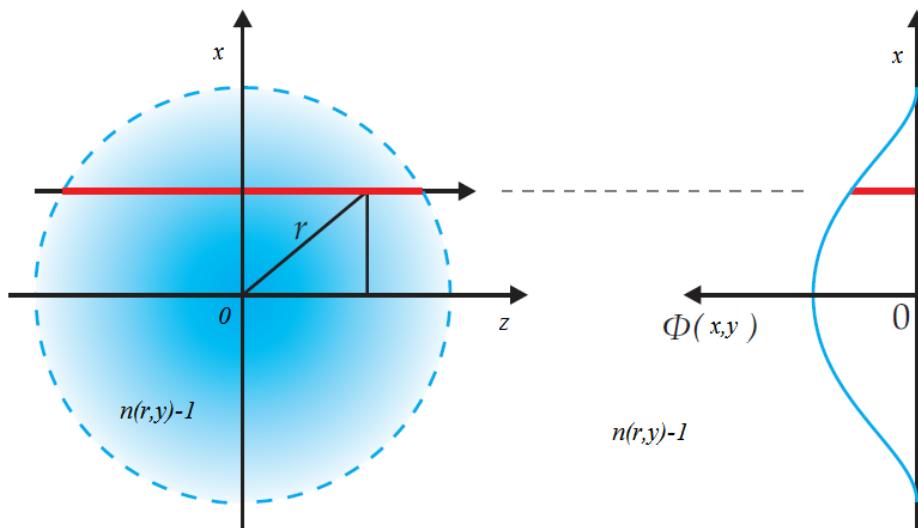


Figura 36. Láser al atravesar el jet de gas

El cálculo que permite obtener la cantidad de fase proyectada, en función de la distribución de índice de refracción en el jet a las distintas alturas es:

$$\phi(x, y) = \frac{2\pi}{\lambda} 2 \int_x^{\infty} \frac{n(\lambda, r, y) - 1}{\sqrt{r^2 - x^2}} dr$$

Sin embargo, en este caso, el objetivo es el contrario: obtener la distribución de índice de refracción, por lo que invirtiendo la relación anterior:

$$n(\lambda, r, y) = 1 + \frac{\lambda - 1}{2\pi} \frac{1}{\pi} \int_r^{\infty} \frac{d\phi(x, y)}{dx} \frac{1}{\sqrt{x^2 - r^2}} dx$$

Donde  $\lambda$  es la longitud de onda del láser utilizado,  $\phi(x, y)$  es la imagen de la fase que se obtiene, siendo "y" la dirección longitudinal al jet de gas y "x" la dirección

transversal, y  $n(\lambda, r, y)$  la distribución radial de índice de refracción que se obtiene como resultado.

Esta operación, la inversión de Abel, evalúa una parte de ese cálculo:

$$\frac{-1}{\pi} \int_r^{\infty} \frac{d\phi(x, y)}{dx} \frac{1}{\sqrt{x^2 - r^2}} dx$$

Que permitirá, aplicando después las cuentas oportunas, recuperar el valor del índice de refracción, a partir del cual, eligiendo un modelo adecuado, se puede obtener la densidad del gas radial del gas a cada altura.

Esta operación incluye una derivada, que es un tipo de operación que tiende a aumentar el ruido. Una forma de reducir este efecto, consiste en aumentar el número de píxeles cercanos (número de vecinos) utilizados para el cálculo de la pendiente en cada punto.

Por último, cabe destacar que como se supone simetría cilíndrica para el jet de gas, para obtener buenos resultados se debe simetrizar antes la imagen.

El resultado entonces de aplicar la operación de inversión de Abel, es la obtención de otra curva para cada altura, como se puede observar en la Figura 37.

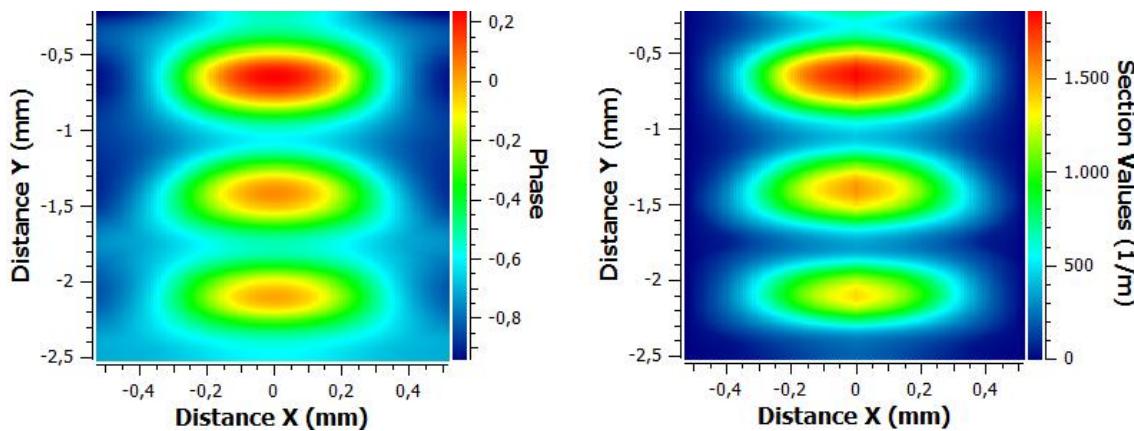


Figura 37. Operación de inversión de Abel

Nótese que mientras que las unidades de la fase son radianes (Adimensional), la inversión de Abel tiene unidades (1/m).

### 3.5.9. Extracción de un rectángulo

Esta operación permite extraer un rectángulo de una zona de la imagen que tenga especial interés, para continuar el procesado usando solamente el rectángulo seleccionado.

El resultado de la aplicación se puede observar en la Figura 38.

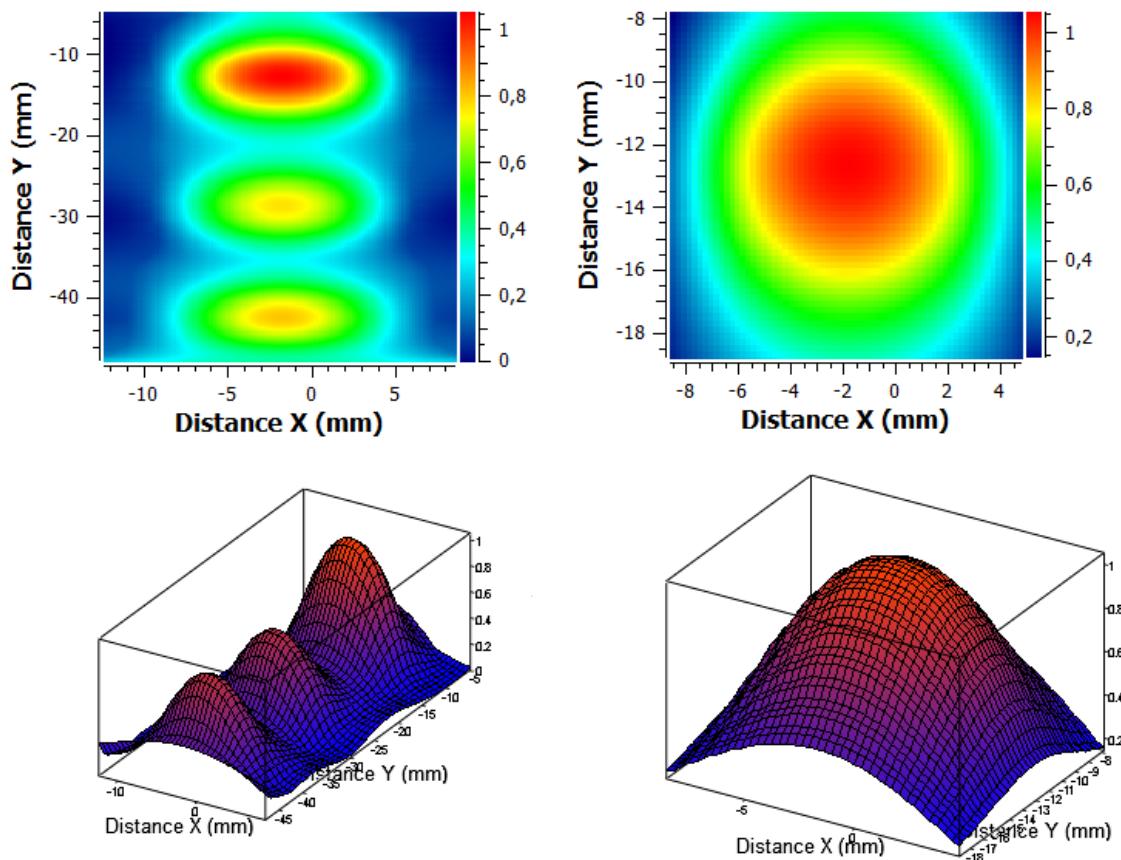


Figura 38. Operación de extracción de rectángulo, antes y después

### 3.5.10. Extracción de una línea

Esta operación permite extraer una recta de una zona de la imagen que tenga especial interés, para continuar el procesado usando solamente la recta seleccionada.

El resultado de la aplicación se puede observar en la Figura 39.

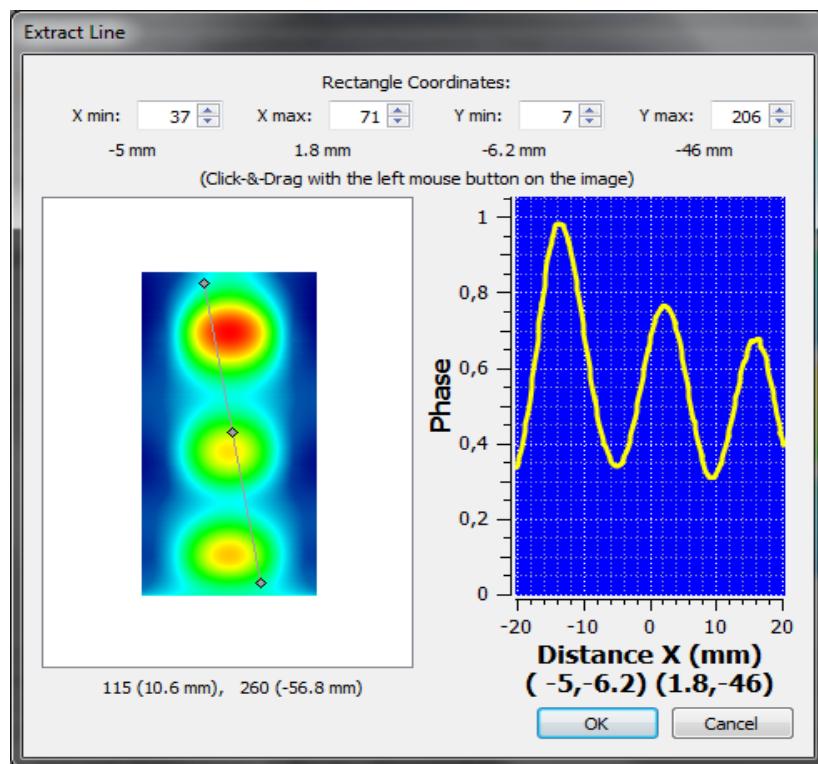


Figura 39. Diálogo de selección de recta

Fuentes: [8][12]

### 3.6. Experimento en el laboratorio

Para acabar, se muestran algunas imágenes del experimento en el laboratorio, y resultados de los obtenidos con los datos tomados.

No se pretende dar una especificación completa del experimento sino, simplemente, enseñar el tipo de montaje utilizado y mostrar resultados reales obtenidos con la aplicación sobre el experimento.

En la Figura 40 se puede observar el esquema del montaje. Las líneas verde y azul, son las dos ramas que participan en el proceso de interferencia: mientras que el rayo azul pasa por el gas, el rayo verde no. Además se observa como la cámara (CCD en el diagrama) se coloca donde los frentes de onda se vuelven a juntar, y por lo tanto hay interferencia.

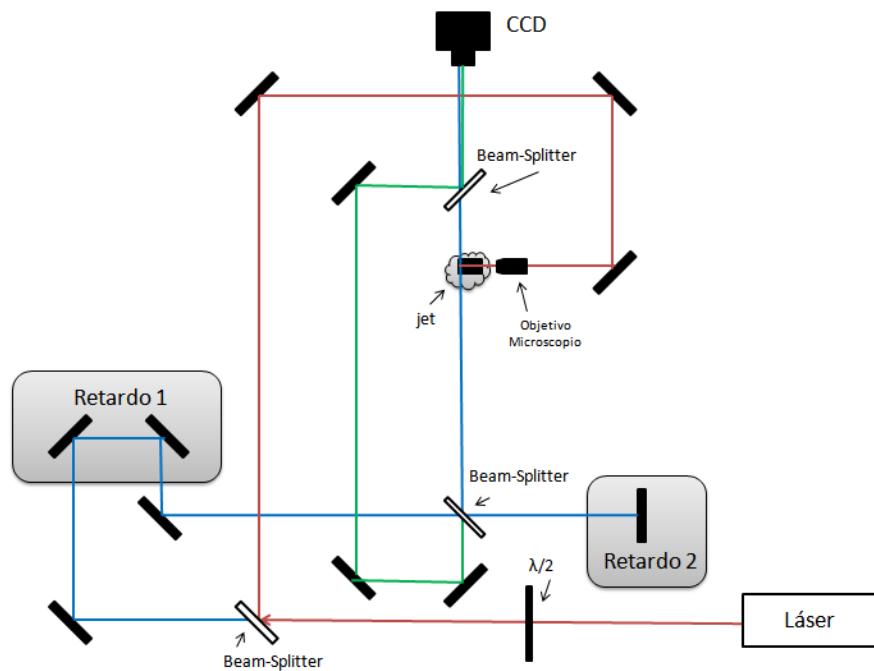


Figura 40. Esquema del montaje en el laboratorio

Y una imagen del mismo montaje se puede contemplar en la Figura 41.

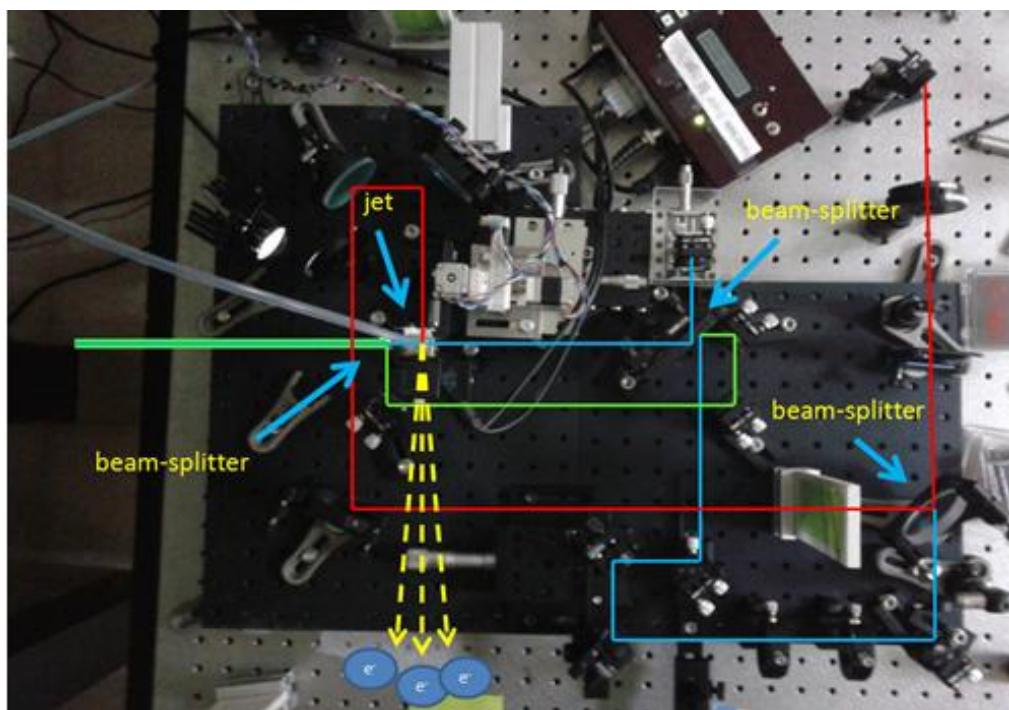


Figura 41. Imagen del montaje en el laboratorio

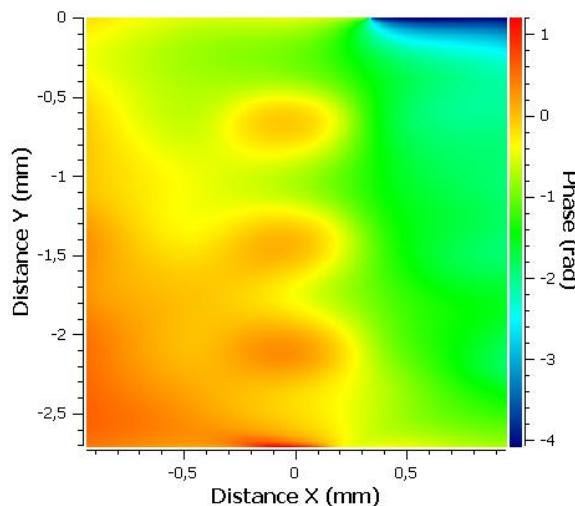
El láser utilizado para iluminar el experimento fue un láser pulsado de Titano-Zafiro de 795 nm de longitud de onda de pico, con una tasa de repetición de 10 Hz, y una duración de pulso de unos 100 fs ( $10^{-13}$  s), con una exposición en la cámara de 99.9ms.

La ventaja del uso de este tipo de láser es que ilumina durante un tiempo tan pequeño que permite observar con mucha nitidez la imagen del jet en un instante determinado, evitando pequeñas vibraciones, que ocurren durante tiempos característicos mayores.

Por otra parte este tipo de láseres pulsados tienen la desventaja de poseer muchas longitudes de onda cercanas al pico, aunque esto no es de mucha importancia en este experimento, ya que basta con considerar la longitud de pico para realizar los cálculos.

El jet de gas también es pulsado, con una presión de 7 bar, y abriendo la válvula a una tasa de 10 veces por segundo coincidiendo con el láser. Uno de los experimentos realizados, y mostrado en uno de los proyectos de ejemplo, consiste en variar gradualmente el retardo entre la apertura de la válvula y la iluminación con el láser, consiguiendo así poder ver toda la evolución del jet de gas desde que se abre hasta que se cierra, tomando y procesando una imagen con cada retardo, y observando después la animación generada al juntar todos los gráficos.

Por último, veamos los resultados obtenidos con el programa en las siguientes ilustraciones para el cálculo del índice de refracción.



*Figura 42. Imagen del cambio de fase sin procesar al pasar por el jet de gas*

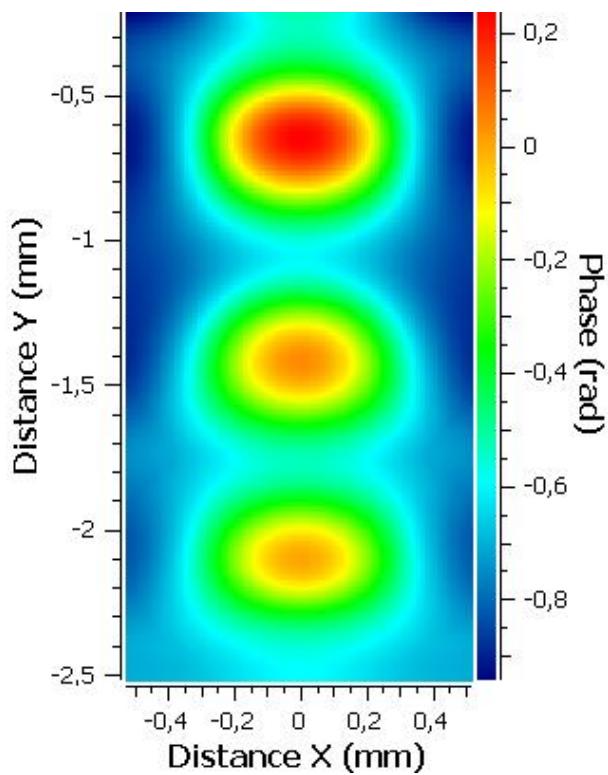


Figura 43. Imagen del mismo cambio de fase, procesado, y simetrizado

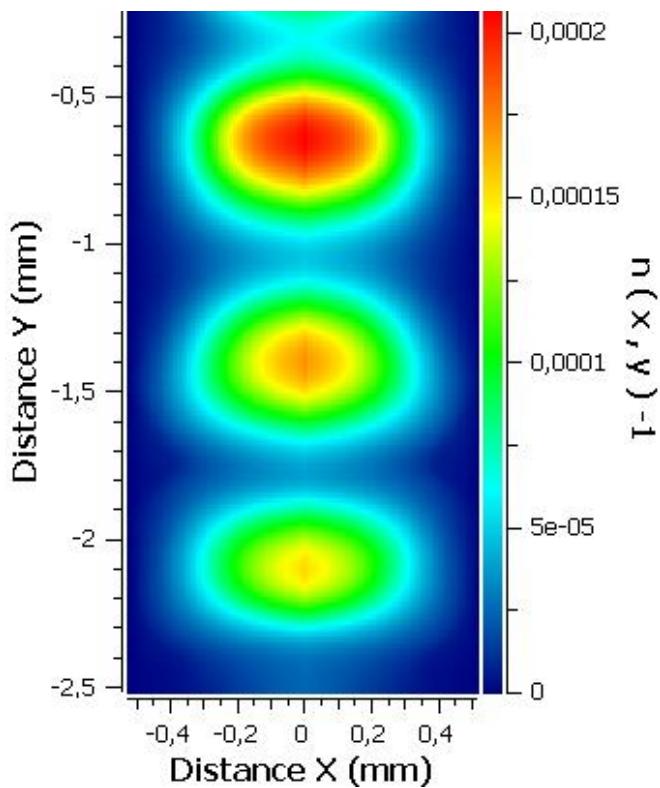
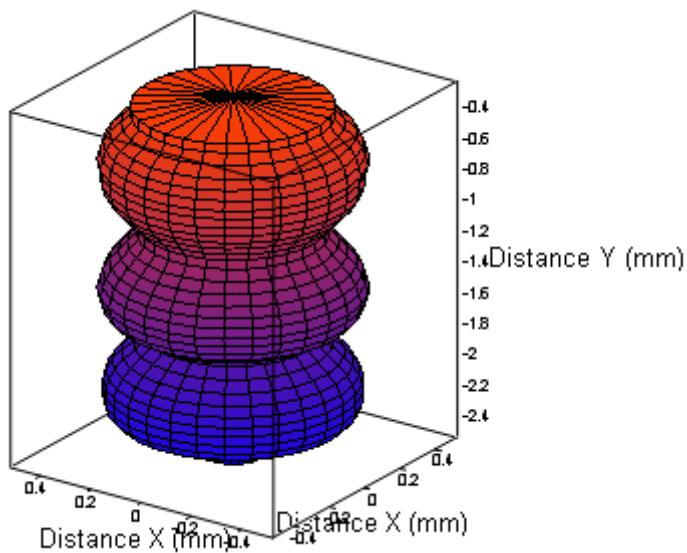


Figura 44. Valores del índice de refracción a cada altura y distancia al centro



*Figura 45. Reconstrucción del jet de gas*

En la Figura 45 se puede observar una reconstrucción del jet de gas generada con el gráfico paramétrico de la aplicación, mostrando la superficie con índice de refracción constante e igual a  $n=2 \cdot 10^{-5} + 1$ .

Y ya para terminar, una curiosidad: los máximos que se pueden observar en las imágenes, son conocidos como “diamantes de choque”, y son zonas de alta presión características de la dinámica de un jet supersónico en el aire, como en los motores a reacción.



*Figura 46. Diamantes de choque en un motor a reacción*



*Figura 47. Diamantes de choque en un caza F-16*

Fuentes: [7][8]



## 4. Técnicas y herramientas

En esta sección se muestran brevemente las diferentes técnicas y herramientas utilizadas para la implementación del proyecto.

Consiste en una breve descripción de cada una de ellas, indicando para qué se han utilizado y el motivo de su uso. Se pueden encontrar más datos específicos del uso de estas herramientas, referencias a las webs, etc. en el Anexo IV.

Se recomienda especialmente la lectura del apartado 4.6 acerca del procesamiento en paralelo con CUDA.

### 4.1. Programación orientada a objetos: C++



Figura 48. Logo de C++

C++ es un lenguaje de programación que surgió a partir del lenguaje C a mediados de los años 80. Se caracteriza por ser multi-paradigma abarcando tanto programación estructurada como programación orientada a objetos.

Hubo varios motivos que hicieron elegir este lenguaje para la implementación del proyecto:

- Es un lenguaje de mucho más bajo nivel comparado con otras opciones como pueden ser Java o Phyton, y permite trabajar con punteros. Siendo el resultado una aplicación a la que se le va a exigir cierto rendimiento, era clave poder controlar toda la gestión de memoria “byte a byte” en las partes más delicadas del programa, para evitar copias de objetos innecesarias y aprovechar al máximo las estructuras ya cargadas en memoria.

- Para el procesamiento en paralelo, se requería utilizar la herramienta CUDA de NVIDIA cuyo código en paralelo se debe escribir en C, que se puede integrar sin problemas con C++.
- A pesar de las características de bajo nivel de C++, gracias a que sigue un paradigma orientado a objetos, es posible utilizar bibliotecas para la creación de interfaces gráficas de forma transparente a través de los objetos ya existentes, y sin tener que preocuparse por la gestión de memoria u otras características de bajo nivel en esos casos.

#### **4.2. Qt y Microsoft Visual C++ 2008 Express Edition**

Una vez fijado el lenguaje de programación, se buscó una biblioteca y entorno de desarrollo de plataforma cruzada ya que, aunque en un primer momento, sólo se pretende construir la aplicación para sistemas Windows, era atractiva la idea de dejar la puerta abierta a otras plataformas.

Para ello se eligió Qt SDK junto con Qt Creator[13].



*Figura 49. Logo de Qt*

El único problema de esto se debió a que la herramienta CUDA, bajo sistemas Windows, sólo tiene soporte para el enlazador de Microsoft Visual Studio.

La solución que se buscó fue el uso del compilador gratuito Microsoft Visual C++ 2008 Express Edition[14], en el entorno de Qt Creator y con la biblioteca de Qt. No obstante como esto no está soportado oficialmente, no estaba garantizado, y de hecho, fue necesario el trabajo de varios días para lograr integrar CUDA con Qt y el compilador/enlazador de Visual Studio, hasta que se logró generar el fichero ".pri"

correspondiente para que *qmake* generara el fichero makefile adecuado incluyendo la etapa de compilación con el compilador de CUDA y el enlazado final.

Por otra parte, Qt SDK se ha utilizado para la creación de toda la interfaz gráfica, así como la programación multihilo, y en general toda la comunicación con el sistema operativo. Esto permite que tanto la programación multihilo, como la gestión de procesos externos, el manejo de archivos y directorios, etc. no tengan dependencia del sistema operativo en el código.

### 4.3. *Bibliotecas de gráficos*

Son las bibliotecas usadas para la representación gráfica.

#### 4.3.1. Biblioteca de gráficos QwtPlot

Es una biblioteca de representación gráfica multiplataforma para Qt [15].

La actual versión de QwtPlot, la 6.x, es compatible con versiones de Qt superiores a la 4.4, y puede ser usada en cualquier entorno en el que funcione Qt acogiéndose, al igual que Qt, a la licencia Pública General Reducida de GNU (LGPL).

Con ella se han implementado las clases Graph1D y SpectrogramPlot, obteniendo los resultados que se pueden observar en la Figura 50.

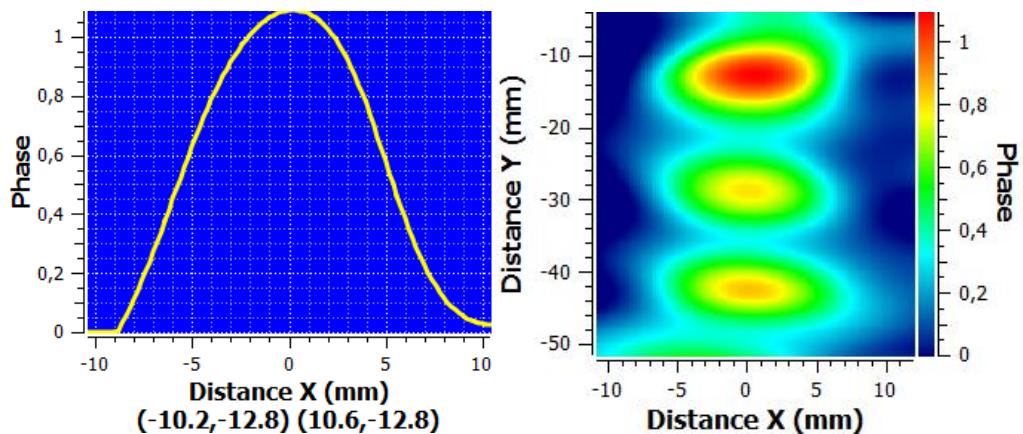


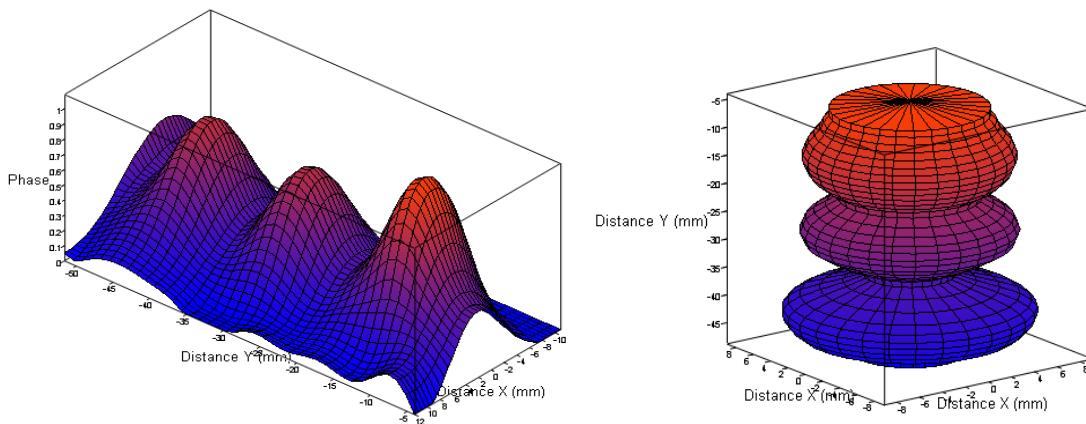
Figura 50. Gráficos Graph1D y SpectrogramPlot respectivamente

#### 4.3.2. Biblioteca de gráficos QwtPlot3D

Es una biblioteca también de representación gráfica multiplataforma para Qt pero que, en este caso, permite representar objetos tridimensionales. Se encuentra en la versión 0.2.7 y puede usarse de forma libre [16].

Su principal desventaja es la inexistencia de documentación apropiada: a pesar de existir, está muy incompleta, por lo que en ocasiones fue necesario leerse el propio código fuente de la biblioteca para lograr algunas características. No obstante constituyó la mejor opción multiplataforma para la representación 3D.

Con ella se han implementado las clases `Surface3DPlot` y `ParametricSurface3DPlot`, con los resultados que se pueden observar en la Figura 51.



*Figura 51. Surface3DPlot y ParametricSurface3DPlot respectivamente*

#### 4.4. ***Biblioteca de compresión QuaZIP***

Es una biblioteca de compresión de código abierto también para Qt, multiplataforma, y basada en ZLib, que se ha utilizado para la compresión de una serie de ficheros en un archivo ZIP, y viceversa [17].

#### 4.5. API *μEye*



Figura 52. Logo de uEye®

Es la biblioteca utilizada para la programación de las cámaras de IDS-Imaging disponibles en el laboratorio. Está escrita en C, y también es multiplataforma [18].

#### 4.6. Procesamiento en paralelo: CUDA

Esta herramienta, CUDA (Compute Unified Device Architecture), de NVIDIA, permite el procesamiento en paralelo con el multiprocesador de la GPU (Graphics Processing Unit) en vez de con la CPU, lo que supone un enorme incremento de la velocidad a la hora de procesar imágenes [19].



Figura 53. Logo de NVIDIA CUDA®

Para desarrollar con esta herramienta es necesario:

- Tener una tarjeta gráfica compatible con CUDA [20]. En este caso se ha dispuesto de una GeForce GTX 550 Ti, y de una GeForce GT 630M.
- Instalar el kit de desarrollo [21]. Para el desarrollo del proyecto se utilizó la versión 4.2 del kit.
- Tener actualizado el driver de la tarjeta de forma que sea compatible con CUDA.

Para ejecutar una aplicación con CUDA simplemente es necesario:

- Tener una tarjeta gráfica compatible con CUDA.
- Tener actualizado el driver de la tarjeta de forma que sea compatible con CUDA.

En los siguientes apartados se describirán los aspectos importantes del uso de esta herramienta. Para más información consúltese la página oficial de CUDA [19] y el libro “CUDA by example” de Jason Sanders y Edward Kandrot [22].

Fuentes: [7][19][22][23][24]

#### 4.6.1. Arquitectura

En la terminología CUDA, siempre se habla de dos partes:

- *Host*: es el cliente de los cálculos, y consiste en la CPU, junto con la memoria principal.
- Dispositivo o *device*: es quién realiza los cálculos, la tarjeta gráfica. Consiste en la GPU, junto con la memoria de la GPU.

En primer lugar, veamos cómo se pasa a trabajar con la GPU desde un programa que se está ejecutando en la CPU, utilizando el ejemplo de la secuencia habitual cuando se programa con CUDA mostrado en la Figura 54.

1. En primer lugar, se debe copiar los datos origen que se quieran utilizar desde la memoria principal a la memoria de la tarjeta gráfica, desde el *host* al dispositivo.
2. En segundo lugar se envía a la GPU el código que tiene que ejecutar en paralelo.
3. Se ejecuta el código en paralelo en el multiprocesador de la GPU.
4. Se copian de nuevo los resultados desde la memoria de la GPU a la memoria principal, desde el dispositivo hasta el *host*.

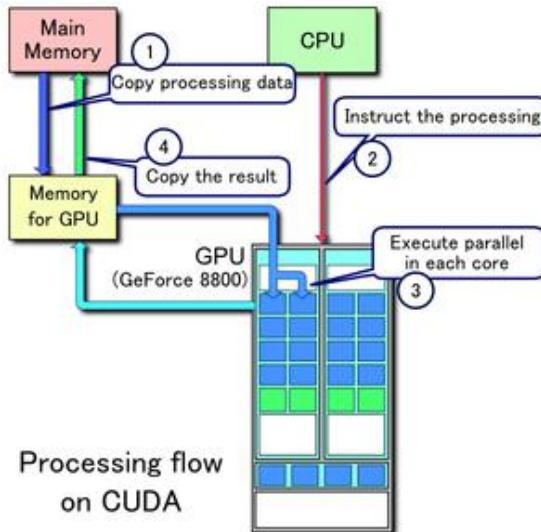


Figura 54. Arquitectura de CUDA junto con la CPU

Y ahora, pasemos a la segunda parte, que es la arquitectura de la GPU.

Cada tarjeta está constituida de uno o varios multi-procesadores o MPs (Multi Processors).

Cada uno de estos MPs posee un cierto número de flujos de procesamiento llamados SP (Stream Processors), pero más conocidos como núcleos CUDA, o *CUDA cores*, pudiendo ejecutar cada uno de ellos instrucciones de forma paralela.

Uno de los parámetros clave en la especificación de una tarjeta compatible con CUDA, es por tanto el número de núcleos CUDA.

Dado que ese número varía de una tarjeta a otra, existe una forma de repartir el procesamiento que es independiente del número de núcleos que posea una tarjeta, siguiendo la distribución que se puede observar en la Figura 55.

- **Hilo o Thread:** Constituye la ejecución de un fragmento de código. Cada hilo posee tres índices que lo localizan dentro del bloque.
- **Bloque o Block:** Un bloque representa un núcleo CUDA de los anteriormente mencionados. Consta de un grupo de hilos, colocados en una disposición tridimensional, pudiendo típicamente contener hasta una cantidad de 512x512x64 hilos. Todos los hilos de un bloque se ejecutan en el mismo MP, y

además en grupos de típicamente 32 (Warp Size) de forma no necesariamente secuencial. Por ello, no existe ninguna garantía del orden de ejecución de los hilos, a no ser que se usen herramientas de sincronización entre ellos. Cada bloque tiene dos índices que lo localizan dentro de la rejilla.

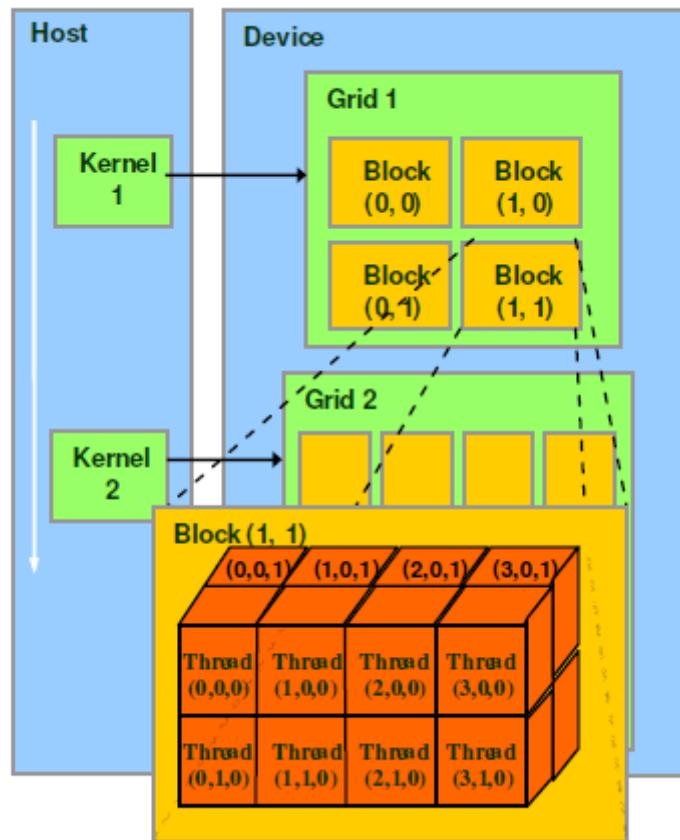


Figura 55. Distribución del procesamiento con CUDA

- **Rejilla o grid:** Es una distribución de bloques, organizados en una red bidimensional, de típicamente un máximo de 65535x65535. Dado que cada bloque representa la ejecución en un núcleo, la ejecución de toda la rejilla se distribuirá dinámicamente entre todos los MPs disponibles, ejecutando uno o varios bloques en cada MP de forma paralela o secuencial. Por ello no será posible la sincronización entre bloques.
- **Fragmento de código o kernel:** Es la unidad de código que se ejecuta en la GPU. Cada vez que se ejecute, se creará una rejilla y una distribución de hilos que

aseguren el procesamiento completo. Nótese que el fragmento de código será compartido por todos los hilos, con distintos índices para cada uno.

En este tipo de programación, es responsabilidad del programador distribuir las tareas de forma adecuada entre los distintos bloques e hilos. Ése proceso se denomina paralelización del código, y requiere un análisis de la tarea a realizar, para su correcta ejecución en paralelo y optimización.

#### **4.6.2. Programación**

La programación para CUDA se realiza en lenguaje C, ligeramente extendido para soportar el procesamiento en paralelo.

Es necesaria la creación de unos archivos con extensión “.cu”, que se compilen con el compilador de CUDA.

La clave del manejo de los datos en CUDA es el uso de punteros. Mientras que habitualmente todos los punteros apuntan a memoria principal, en este caso, los punteros también pueden apuntar a memoria de la GPU. No obstante, estos dos tipos de punteros son exactamente iguales formalmente, por lo que es importante controlar a qué tipo de memoria está apuntando cada puntero en cada momento.

Dichos archivos contienen dos tipos de funciones:

- Las funciones de acceso a los *kernels*: Son funciones habituales de C que se ejecutan en la CPU. Desde ellas se realizan las gestiones de memoria correspondientes para transferir la memoria del *host* al dispositivo y viceversa, y se lanzan los *kernels*, indicando la distribución de rejilla que se debe ejecutar para el procesamiento en paralelo. Desde estas funciones sólo se puede acceder a los punteros que apuntan a la memoria principal, mientras que los que apuntan a la memoria de la GPU sólo se pueden usar para reservar/copiar/liberar memoria siempre a través de las funciones de la API.
- Los *kernels*: Son los fragmentos de código que realmente se ejecutan en el procesador, es el mismo código para todos los hilos que se ejecuten, salvo por

ciertas variables globales, los índices, que sirven para determinar qué hilo se está ejecutando. Desde estas funciones sólo se puede acceder a los punteros que apunten a memoria de la GPU, y además, considerando que las zonas de memoria apuntadas, podrían ser accedidas por otros hilos ejecutándose a la vez, lo que hace necesario tener en cuenta la concurrencia.

El lanzamiento de los *kernels* es la parte cuya sintaxis varía respecto de la de C, pues hay que indicar el tamaño de la rejilla. La llamada genérica es:

```
dim3 threads(X,Y,Z);
dim3 blocks(M,N);
KernelCode <<< blocks, threads >>>((type1*) pt1,(type2*) pt2...,);
```

De esa forma se está creando una rejilla bidimensional de M x N bloques, con un distribución tridimensional en cada uno de ellos de X x Y x Z hilos.

Los punteros pt1, pt2..., deben estar apuntando a memoria reservada en la GPU, por lo que normalmente también es necesario el paso de argumentos por valor con los tamaños de las zonas de memoria.

Después, en la ejecución de los kernels, tendremos acceso a las siguientes estructuras globales, que ayudaran a identificar el hilo que se está ejecutando

gridDim.x	Número de bloques de ancho de la rejilla. M en el ejemplo.
gridDim.y	Número de bloques de alto de la rejilla. N en el ejemplo.
blockIdx.x	Primer índice del bloque ejecutándose actualmente. De 0 a M-1.
blockIdx.y	Segundo índice del bloque ejecutándose actualmente. De 0 a N-1.
blockDim.x	Número de hilos de ancho de cada bloque. X en el ejemplo.
blockDim.y	Número de hilos de alto de cada bloque. Y en el ejemplo.
blockDim.z	Número de hilos de fondo de cada bloque. Z en el ejemplo.
threadIdx.x	Primer índice del hilo ejecutándose actualmente. De 0 a X-1.
threadIdx.y	Segundo índice del hilo ejecutándose actualmente. De 0 a Y-1.
threadIdx.z	Tercer índice del hilo ejecutándose actualmente. De 0 a Z-1.

#### 4.6.3. Ejemplo de uso

En este apartado veremos un ejemplo práctico de uso en un caso muy sencillo, como es, para una matriz de números complejos de entrada, calcular una matriz con la fase de cada número a la salida.

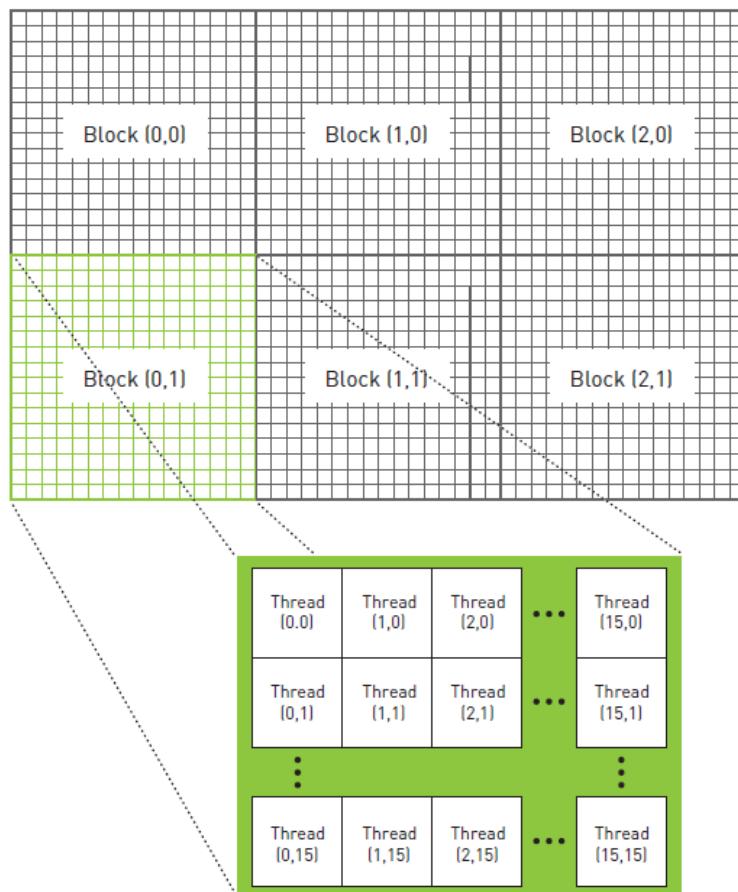


Figura 56. Distribución del trabajo para el procesado pixel a pixel

Nuestro objetivo es procesar la matriz distribuyendo las tareas según la Figura 56.

El tipo de datos complejo creado es:

```
typedef struct {
    float x;      /*< Parte real.*/
    float y;      /*< Parte imaginaria.*/
} complex;
```

Y supongamos que tenemos un puntero que apunta a una zona de memoria con la matriz de entrada con dimensiones *width* x *height* de estructuras tipo *complex* alineadas fila tras fila en memoria principal, el *host*:

```
complex * sourceHost;
```

En primer lugar reservamos memoria en la GPU, el dispositivo, para los datos de entrada y los datos de salida:

```
complex * sourceDevice;
float * destDevice;
cudaMalloc((void**)& sourceDevice, sizeof(complex)*width*height);
cudaMalloc((void**)& destDevice, sizeof(float)*width*height);
```

A continuación copiamos los datos de entrada del *host* al dispositivo en la memoria reservada para ellos:

```
cudaMemcpy(sourceDevice, hostDevice,
           sizeof(complex)*height*width, cudaMemcpyHostToDevice);
```

Después definimos la rejilla, con 16 x 16 hilos por núcleo, y tantos bloques como necesitemos, redondeando por arriba y lanzamos el *kernel*:

```
int blocksX=(15+width)/16;
int blocksY=(15+height)/16;
dim3 blocks(blocksX,blocksY);
dim3 threads(16,16);
CalculatePhase<<<blocks,threads>>>
    (sourceDevice,destDevice,height,width);
```

El código del kernel es el siguiente, donde cada hilo se encarga de procesar un pixel de la matriz, escribiendo la fase en la misma posición de la matriz de salida:

```
__global__ void CalculatePhase
    (complex * source, float * dest, int sizeX , int sizeY){

    float real,imag;
    int x = threadIdx.x+blockIdx.x*blockDim.x;
    int y = threadIdx.y+blockIdx.y*blockDim.y;

    if(x<sizeX && y<sizeY){
        real= source[x+y*sizeX].x;
        imag= source[x+y*sizeX].y;
        dest[x+y*sizeX]=atan2f(imag,real);
    }
}
```

A continuación reservamos memoria en la CPU para el resultado, y copiamos el resultado del dispositivo al *host*:

```
float * destHost;
destHost = (float*) malloc(sizeof(float)*width*height);
cudaMemcpy(destHost, destDevice,
```

```
sizeof(complex) *height*width, cudaMemcpyDeviceToHost);
```

Por último liberamos la memoria en el dispositivo:

```
cudaFree(destDevice);
cudaFree(sourceDevice);
```

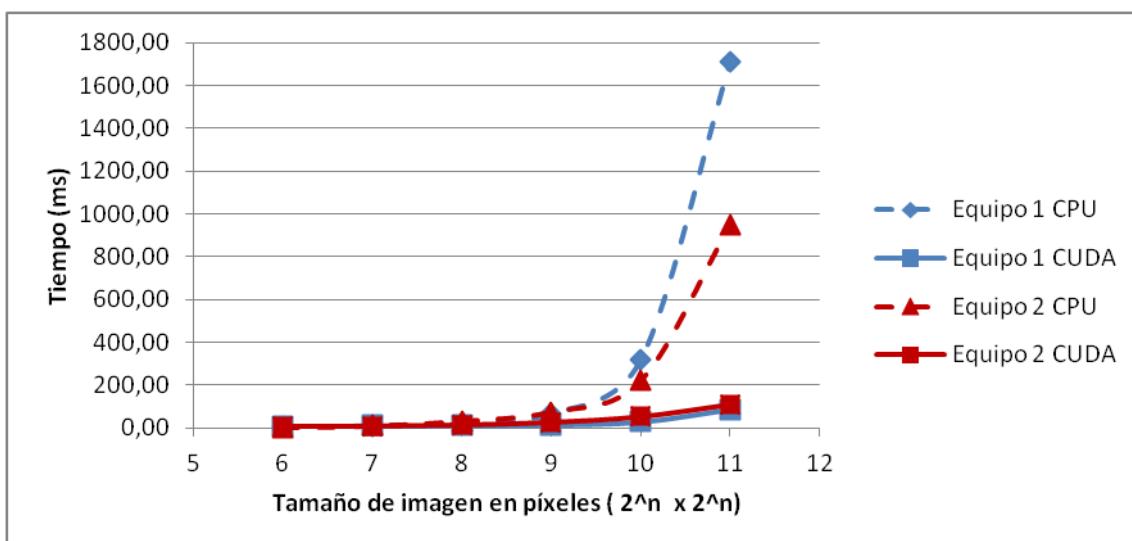
#### 4.6.4. Biblioteca cuFFT

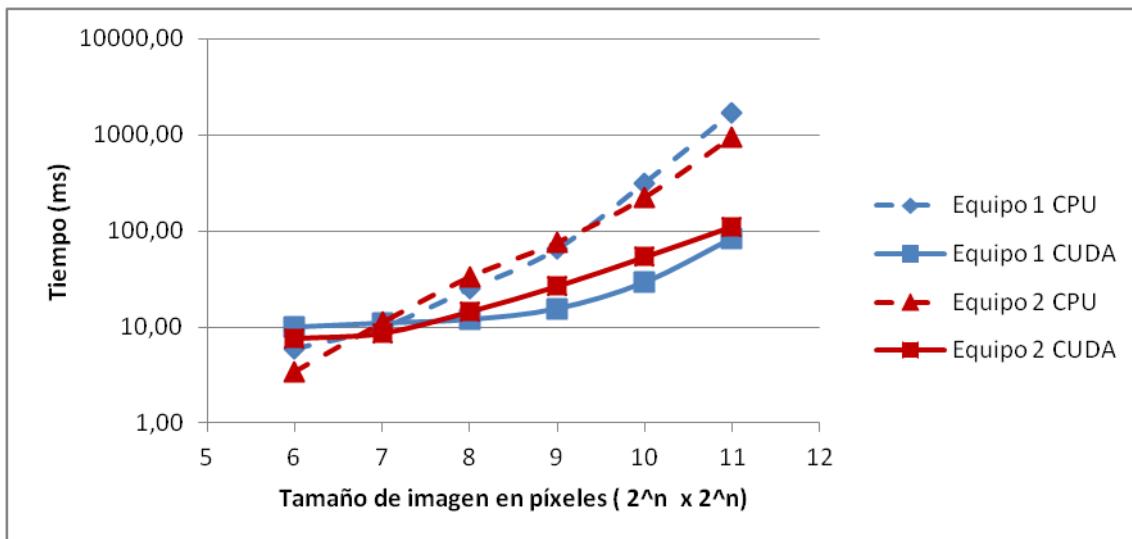
La biblioteca cuFFT es una biblioteca proporcionada por NVIDIA junto con la SDK de CUDA para el cálculo de la transformada de Fourier numérica con el algoritmo FFT de forma óptima con la tarjeta gráfica.

Es la biblioteca que se ha utilizado para calcular la transformada de Fourier compleja en el caso de la implementación con CUDA, siendo esta la única operación que se realiza mediante biblioteca, ya que para el resto se han implementado todos los *kernel*.

#### 4.6.5. Comparativa de velocidad: CPU vs CUDA

En la Figura 57 se pueden observar las diferencias de tiempos en el procesado completo de una imagen, comparando el tiempo medio de cálculo con la CPU frente al tiempo de cálculo con la herramienta CUDA para dos equipos distintos.





Tamaño		Tiempos (ms)			
Dimensiones	Potencia de 2	E1 CPU	E1 CUDA	E2 CPU	E2 CUDA
64x64	6	6,00	10,00	3,45	7,67
128x128	7	10,00	11,00	11,17	8,67
256x256	8	25,00	12,00	33,19	14,50
512x512	9	65,13	15,55	76,19	26,59
1024x1024	10	316,88	29,32	222,96	53,53
2048x2048	11	1709,21	84,60	949,35	110,90

Figura 57. Comparativa entre CPU y CUDA, en escala natural y logarítmica

Las características de los dos equipos utilizados son:

- Equipo 1: Equipo de escritorio con Windows XP:
  - Intel® Core 2 Duo E6750 a 2,66 GHz.
  - 1,5 GB de memoria RAM DDR 2.
  - Tarjeta gráfica NVIDIA GeForce GTX 550 Ti con 192 núcleos CUDA y una 1GB de memoria dedicada.
- Equipo 2: Equipo portátil con Windows 7:
  - Intel® Core i5 2450M a 2.50GHz (3.1GHz Turbo Boost).
  - 8 GB de memoria RAM DDR3.
  - Tarjeta gráfica NVIDIA GeForce GT 630M con 96 núcleos CUDA y 2GB de memoria dedicada.

Un detalle que se puede observar es que durante el primer tramo, para tamaños pequeños, la CPU puede llegar incluso a ser más rápida que CUDA, y que de hecho, el tiempo que necesita CUDA, es prácticamente independiente del tamaño. Esto se debe, a que en los tiempos totales, se ha incluido también el tiempo empleado en la transferencia entre la memoria principal y la memoria de la GPU, que son los que en este caso están limitando inferiormente el tiempo de cálculo.

Otro detalle que cabe destacar es la influencia del número de núcleos en el cálculo con CUDA. Para imágenes pequeñas, el equipo 2, con menos núcleos de CUDA, supera al equipo 1, sin embargo, en ese tramo, estábamos limitados por la transferencia de memoria, y la memoria principal segundo equipo es más rápida (DDR3 frente a DDR2) y el procesador y sistema operativo más moderno. Sin embargo, para los últimos tres tamaños de imagen, el equipo 1, con el doble de núcleos CUDA, presenta tiempos un 42%, 45% y 24% más rápidos, respectivamente que el equipo 2.

#### **4.7. Archivos INI**

El formato INI consiste en un simple archivo de texto en ASCII, utilizado como estándar para almacenar y leer parámetros de configuración [7].

En Qt se pueden manejar a través de la clase QSettings, y simplemente consisten en un archivo con tres tipos de elementos, cada cual en una línea distinta del fichero:

- Secciones: Para agrupar parámetros relacionados.  
Formato: “[nombreSección]”.
- Valores: Para almacenar los valores de los parámetros.  
Formato: “nombreParámetro=valorParámetro”.
- Comentarios: Para incluir información adicional.  
Formato: “; Comentario”.

Este tipo de archivos tienen la ventaja de ser extensibles, es decir, que aunque se añadan parámetros en una nueva versión de la aplicación, los archivos generados con la versión anterior pueden ser utilizados y actualizados de forma transparente.

En el proyecto se utilizan para almacenar en memoria secundaria los parámetros más sencillos de la aplicación, como son las preferencias del programa, los parámetros de la cámara, los archivos de máscara, las opciones de visualización...

#### **4.8. XML**

Los archivos .INI, aunque extensibles, no permiten almacenar estructuras más complejas como listas. Aquí es donde entra en juego XML.

XML (Extensible Markup Language), es un metalenguaje estándar que permite crear archivos de texto ASCII con una estructura de árbol para almacenar variables [7].

Al igual que los archivos INI, tiene también la gran ventaja de ser extensible, aunque la sintaxis es algo más compleja, por lo que no se entrará en detalle hasta que no se describan más adelante los ficheros.

Existen dos modos de acceso a los ficheros XML:

- DOM (DocumentObjectModel): Crea una estructura de objetos con el árbol contenido en el archivo XML.
- SAX (Simple API for XML): Va procesando el documento de manera asíncrona y generando eventos cada vez que se cargan elementos nuevos. Es ligeramente más rápido que DOM, dado que es de más bajo nivel.

En la aplicación creada se utiliza XML para el guardado de la lista de algoritmos y de la lista de imágenes, incluyendo en este caso una referencia a un archivo binario con los datos de las imágenes, además dada su sencillez de uso se ha optado por DOM, ya que el tiempo al cargar/guardar un elemento no es un parámetro extremadamente sensible en este caso.

El tratamiento de los archivos XML con el modo de acceso DOM se hace en Qt a través de la clase QDomDocument.

Es importante el hecho de que los archivos XML, al contrario que los archivos binarios, pueden ser también comprendidos como texto plano por el ser humano, y por tanto pueden ser creados y modificados, no sólo por el programa que los genera, sino también por el usuario, conocida su estructura.

#### **4.9. *MPlayer: MEncoder***



*Figura 58. Logo de MPlayer*

Es una herramienta de MPlayer utilizada para crear los archivos de vídeo en formato AVI con las secuencias de imágenes bajo licencia “GNU General Public License” [25].

Consiste en un ejecutable que acompaña a la aplicación, y que es llamado por línea de comandos a través de la clase de Qt QProcess.

Un ejemplo de comando que hay que ejecutar es:

```
mencoder.exe mf://*.bmp -mf w=640:h=480:fps=24:type=bmp -ovc lavc lavcopts vcodec=mpeg4:vbitrate=2000 -oac copy -o salida.avi;
```

Esta aplicación es multiplataforma aunque, obviamente, es necesario cambiar el ejecutable.

#### **4.10. *ImageMagick: Converter***

Es una herramienta de ImageMagick utilizada para crear las animaciones en formato GIF a partir de una secuencia de imágenes bajo licencia “Apache 2.0 license” [26].



Figura 59. Logo de ImageMagick

Consiste también en un ejecutable que acompaña a la aplicación, y que es llamado por línea de comandos a través de la clase de Qt QProcess, siendo un ejemplo de comando:

```
convert.exe -delay 100 -loop 0 *.bmp salida.gif
```

Esta aplicación también es multiplataforma.

#### **4.11. RealWorld Icon Editor**

RealWorld Icon Editor es una herramienta de Real World Graphics, disponible en la web [27].

Es el programa utilizado para diseñar los iconos de la aplicación. La mayor parte han sido creados exclusivamente para el programa, desde cero, o a partir de elementos obtenidos de las librerías libres de iconos.

#### **4.12. Install Creator Pro**

Es una herramienta de ClickTeam, para la generación de instaladores en Windows [28].

Se ha utilizado para la generación del instalador de la aplicación a partir de los archivos.

Facilita distintas tareas como:

- Creación del instalador con los archivos empaquetados.
- Extracción de los archivos en los directorios deseados.

- Escritura en el registro.
- Creación automática de un desinstalador.

Y todo ello permitiendo una gran personalización del instalador como la muestra de opciones, tipo de instalación, imágenes...

#### ***4.13. Herramientas para la documentación***

En este apartado se indican las herramientas que han sido utilizadas para la generación de la documentación.

- **Microsoft Word 2007 [29]:** Procesador de textos usado para la redacción y maquetación final de la memoria.
- **Doxxygen [30]:** Herramienta libre y gratuita para la documentación del código fuente y en particular las clases. Permite generar dicha documentación en diferentes formatos, habiéndose elegido en este caso el formato web, que se puede encontrar como segunda parte del Anexo IV.
- **REM 1.2.2 [31]:** Herramienta CASE utilizada para la creación de tablas de objetivos, requisitos y casos de uso con el formato adecuado. Creada por Amador Durán Toro, en el contexto del Departamento de Lenguajes y Sistemas Informáticos en la Universidad de Sevilla.
- **Visual Paradigm for UML 8.0 [32]:** Herramienta CASE utilizada para la creación de los diagramas de UML, en este caso diagramas de casos de uso, de clases, de paquetes, de niveles, y de secuencia.
- **Microsoft Paint:** Para el retocado rápido de imágenes.
- **Microsoft Excel 2007 [29]:** Para la generación de diagramas de Gantt y otros gráficos.
- **Adobe Photoshop CS [33]:** Para la creación y edición de imágenes más elaboradas.



## 5. Aspectos relevantes del desarrollo del proyecto

En esta sección se incluyen todos los aspectos más interesantes del desarrollo del proyecto. La mayor parte de la información aquí disponible aparece también en los distintos anexos, aunque en algunos casos el contenido es diferente: mientras que la información en los anexos aparece desde un punto de vista mucho más técnico y riguroso, aquí se pretende hacer llegar al lector todo el conjunto de estrategias que se ha utilizado para el desarrollo del proyecto, dando así una visión más global de éste.

### 5.1. Planificación

Este apartado constituye un resumen de la planificación realizada para el desarrollo del proyecto, que se puede encontrar en el Anexo I.

Desde el comienzo del proyecto en Marzo de 2012 se planteó como objetivo una fecha de finalización para Septiembre de 2012. También desde entonces se supo que la distribución temporal del trabajo, por motivos académicos, iba a ser muy poco homogénea en el tiempo, de forma que la mayor parte del trabajo iba a estar muy desplazada hacia la segunda parte de ese periodo, a partir del 7 de Junio.

Conocido eso, se planteó, en primer lugar, una primera etapa de trabajo en el proyecto de toma de requisitos y diseño del sistema, procurando no dejar nada fuera, y realizando durante todo ese tiempo pruebas con las bibliotecas que iban a ser necesarias, CUDA, la cámara..., sin implementar nada de código definitivo para el proyecto, simplemente haciendo prototipos de uso.

El hito al salir de esta etapa era tener unos requisitos muy bien definidos, un buen esquema del diseño, y haber encontrado y afrontado todos los puntos problemáticos que pudieran surgir y que más tiempo podrían hacer perder al usar APIs o bibliotecas, o configurar el entorno de trabajo.

Esta etapa de trabajo, con 1-2 horas diarias dedicadas al proyecto, compaginadas con las clases, se extendió hasta mediados de mayo, con el comienzo de los exámenes.

El trabajo en el código de la aplicación no comenzó hasta principios del mes de Junio, justo al finalizar los exámenes.

Esta segunda etapa de trabajo, consistió en jornadas completas durante todos los días laborables, de entre 6 y 10 horas, y se extendió hasta las fases finales del proyecto, en Septiembre de 2012.

La dedicación total y exclusiva al proyecto, permitió adquirir una excelente percepción global de todo el proyecto, lo que permitió avanzar muy rápido, primero con la implementación de la aplicación, y más adelante con la redacción definitiva de la memoria y los anexos, sin encontrar apenas trabas, debido al tiempo que se había dedicado en la primera etapa en buscar los puntos problemáticos.

No obstante, el proceso seguido, especialmente durante la implementación que se caracterizó por ser muy incremental, aunque también durante el análisis y diseño, se adapta perfectamente al proceso unificado, siguiendo las siguientes fases

- **Fase de inicio (10/03/2012-04/04/2012):** Se centró en la toma de requisitos y la especificación de casos de uso. Se procuró crear una carta de requisitos lo más completa posible, aunque no se pudo evitar la necesidad de incluir alguno adicional en etapas más avanzadas. Consistió en dos iteraciones de toma y validación de requisitos.
- **Fase de elaboración (05/04/2012-12/05/2012):** Se centró en el análisis completo del problema, terminando de detallar los casos de uso, y en el diseño de los módulos de la aplicación. Además durante ella se realizaron pruebas con las bibliotecas. Constó de 3 iteraciones principales:
  - **Diseño del sistema base:** Diseño de todo el núcleo de la aplicación, las estructuras de almacenamiento, y los módulos de cálculo para realizar las operaciones.
  - **Diseño de la interfaz:** Se crearon bocetos sobre el papel con las ventanas finales que se enseñaron al cliente, aportando éste ideas adicionales.

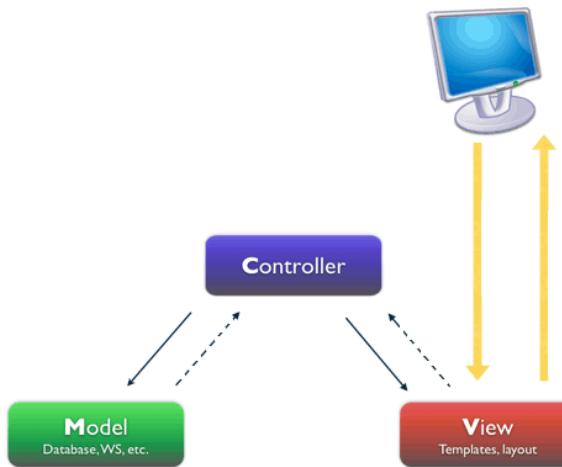
- **Refinamiento del sistema base:** Tras diseñar la interfaz y seguir madurando la idea de la aplicación, surgieron nuevas ideas para el sistema base que modificaron el diseño anterior.
- **Fase de construcción (13/05/2012-31/07/2012):** Durante esta fase se implementó toda la aplicación, salvo cambios menores que se introdujeron a posteriori. Consistió en 4 iteraciones principales:
  - **Búsqueda del entorno de desarrollo:** Se buscó y configuró el entorno de desarrollo adecuado para el proyecto.
  - **Implementación de los elementos del sistema base:** Esta iteración podría incluir sub-iteraciones, ya que los elementos del sistema base se desarrollaron de forma incremental, creando pequeños prototipos desechables de interfaz gráfica para realizar las pruebas de cada módulo.
  - **Puesta de los elementos en conjunto:** Se crearon la interfaz gráfica y el controlador principal centralizando así toda la funcionalidad de la aplicación.
  - **Implementación de la funcionalidad añadida:** Por último se añadió funcionalidad adicional, como el modo “en vivo”, opciones de guardado, y el simulador de cámara.
- **Fase de transición (01/08/2012-15/09/2012):** Al inicio de esta fase se cuenta con una versión beta del proyecto, sobre la que se realizan pruebas, y sobre la que se va añadiendo funcionalidad menor como advertencias de guardado, opciones adicionales para los gráficos, etc. Además, en paralelo, se fue redactando la memoria definitiva, y recopilando todos los diagramas y tablas que ya se habían generado para los anexos.

## **5.2. Arquitectura utilizada**

Este apartado es un brevísimo resumen de su homónimo en el Anexo III, que debería consultarse para más información.

### 5.2.1. Modelo-Vista-Controlador modificado

Para la gestión de la interfaz gráfica se ha utilizado una modificación del modelo vista controlador, que ha permitido eliminar la dependencia entre el modelo y la vista, de la forma que se puede observar en la Figura 60.



*Figura 60. Patrón Modelo-Vista-Controlador modificado*

Las flechas representan el traspase de información entre los distintos elementos del patrón. Si la flecha es continua indica que hay dependencia directa, mientras que la flecha discontinua indica que no hay dependencia directa, es decir, información que se recibe al salir de un método (La clase que implementa el método no tiene porqué conocer a la clase que lo llama), o bien mediante el sistema de señales/*slots* de Qt que permite transferir eventos sin que el emisor de la señal tenga que conocer al receptor.

La principal ventaja, como ya se ha comentado, es la eliminación de la dependencia entre el modelo y la vista. Esto hace mucho más fácil cualquier cambio tanto en el modelo, como en el controlador, como en la vista. Pues el modelo no depende ni de la vista ni del controlador, la vista no depende ni del controlador ni del modelo, y es el controlador el único que depende, por tanto, de la vista y del modelo.

Exactamente, el cometido de cada uno de los componentes, en este patrón modificado, y para el problema concreto, es el siguiente:

- Vista: Será una clase generada automáticamente por el diseñador de Qt. Contiene todos los visualizadores y controles y se encargará de enviar señales cada vez que se produzca una acción por parte del usuario. Un ejemplo en la aplicación es “Ui::MainWindow”.
- Controlador: Existe uno para cada vista. En general, dada esa relación uno a uno con la vista, tendrá su mismo nombre (Aunque en distintos espacios de nombres). Esta clase sí estará directamente codificada por el programador. Se encarga de recibir en sus *slots* todas las señales procedentes de la vista, y en base a ellos modificar el modelo o la vista según proceda. Un ejemplo en la aplicación es “MainWindow”.
- Modelo: Contiene la lógica de cálculo de la aplicación, así como acceso a todos los datos. Recibirá peticiones del controlador. En la aplicación, el modelo contiene a su vez, otro controlador, el controlador central de la aplicación (MainController), a través del que se accede a todos los datos o a otros controladores con acceso a los datos.

Por último, veamos en la Figura 61 un ejemplo con clases reales de la aplicación.

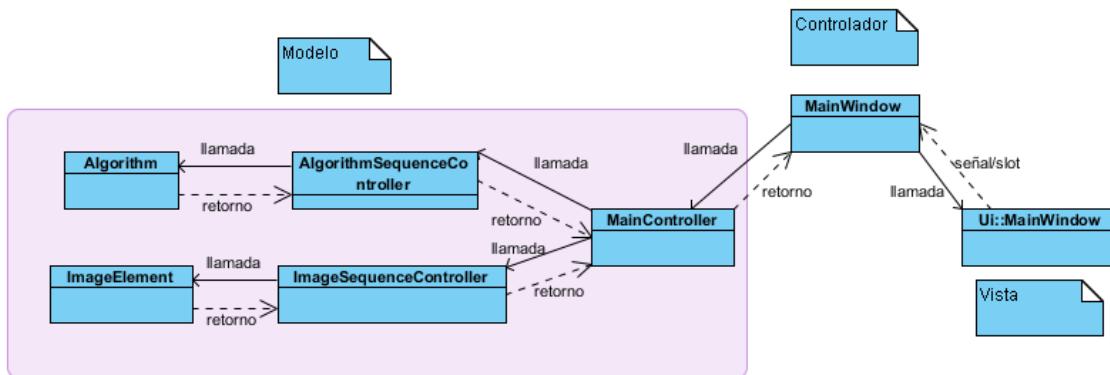


Figura 61. Ejemplo de MVC modificado

### 5.2.2. Vista de capas:

Tal y como se comentó en la planificación, el programa se ha ido desarrollando de forma incremental, creando por separado los módulos encargados de cálculos, de almacenamiento, y de representación gráfica.

Esto ha llevado a una estructura muy modular, con pocas dependencias cruzadas, que se puede observar en la Figura 62.

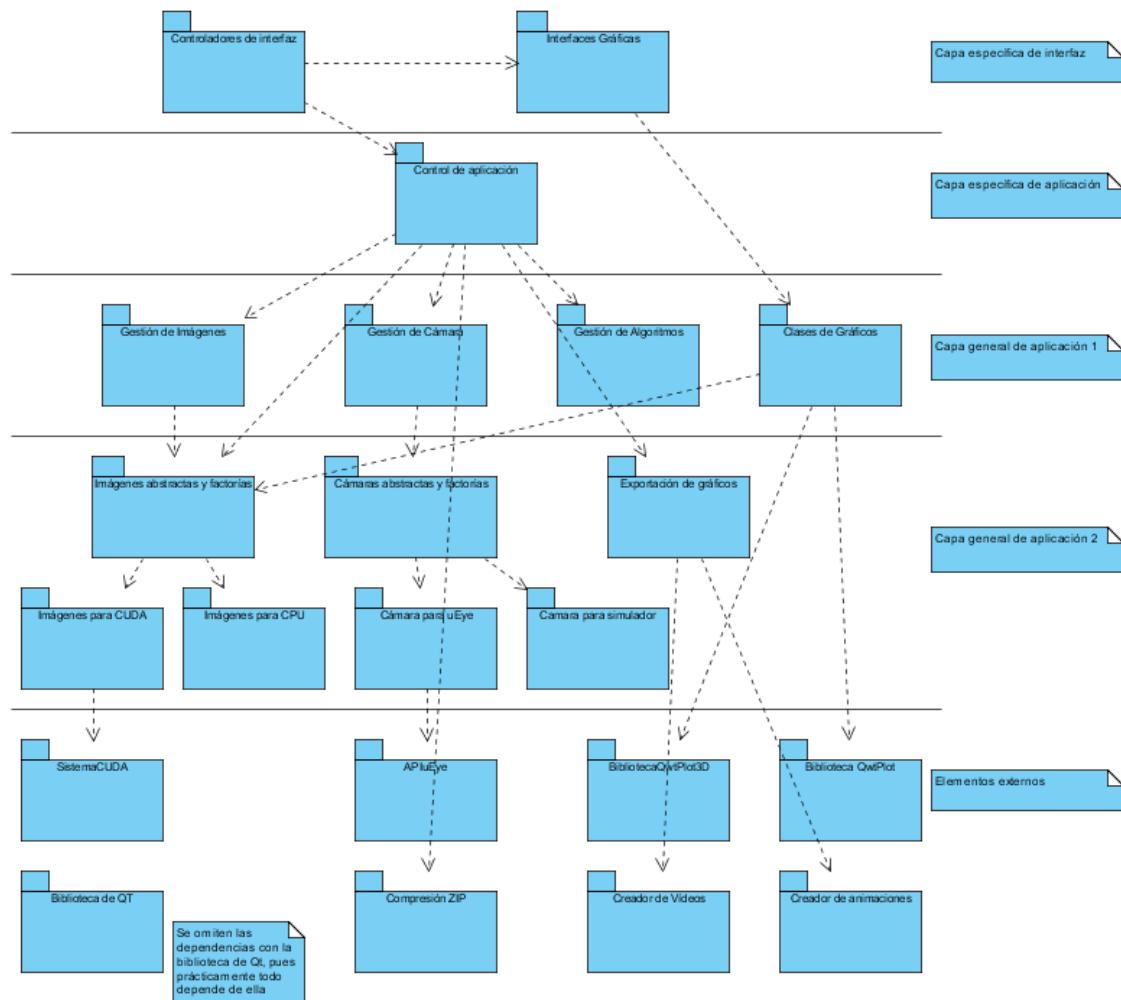


Figura 62. Diagrama de capas del sistema

Tiene especial importancia el paquete “Control de aplicación”, que contiene al controlador principal, y que se encarga de comunicar a todos los módulos, y aísla totalmente las clases relacionadas con la interfaz gráfica de los módulos de almacenamiento y de cálculo, así como a los módulos entre sí. Esto hace que cada módulo, sea muy fácil de reutilizar.

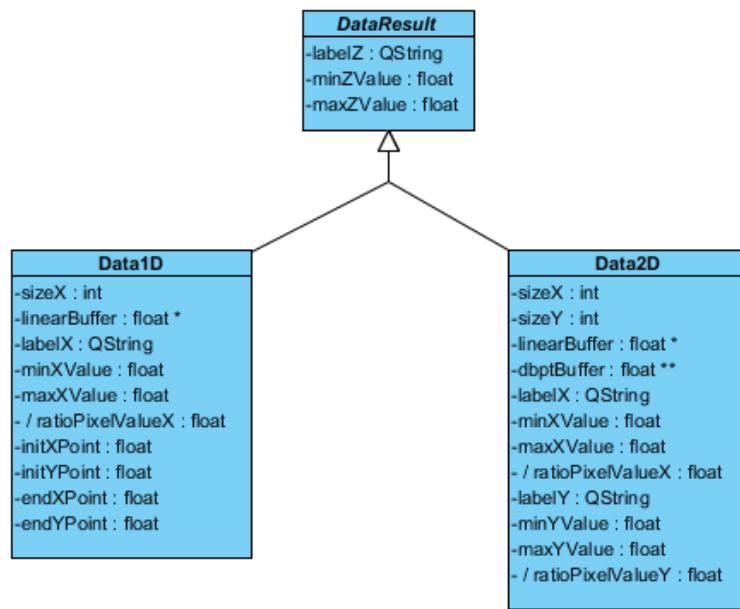
### **5.3. Descripción de la interfaz gráfica**

Este apartado contiene información acerca de la parte visual de la aplicación. Todos los elementos pueden encontrarse también en el Anexo III, explicados con más detalle y de una forma más técnica.

#### **5.3.1. Tipos de gráficos y sus tipos de datos**

Existen dos clases de resultados a los que se pueden llegar en la aplicación:

- Clase Data2D: Datos bidimensionales, una imagen de la fase propiamente dicha, antes o después de ser procesada.
- Clase Data1D: Datos unidimensionales, procedentes de una línea que se ha extraído de la imagen de la fase procesada.



*Figura 63. Diagrama de clases de los objetos de resultados*

Estos dos tipos de datos, se organizan mediante una jerarquía siguiendo el esquema que se puede mostrar en la Figura 63.

Nótese que estas clases sólo se usan para representar resultados, y no para realizar cálculos, por lo que son independientes del modo de cálculo CUDA o CPU.

Continuando con los resultados, pasemos a comentar las clases de representación.

La misión de las clases de representación de gráficos es siempre representar un objeto del tipo Data1D o del tipo Data2D, existiendo en la aplicación cuatro tipos diferentes de gráficos.

Para aprovechar de nuevo las ventajas de la abstracción y la herencia, los 4 tipos de gráficos se distribuyen como los nodos hoja de la jerarquía que se puede observar en la Figura 64.

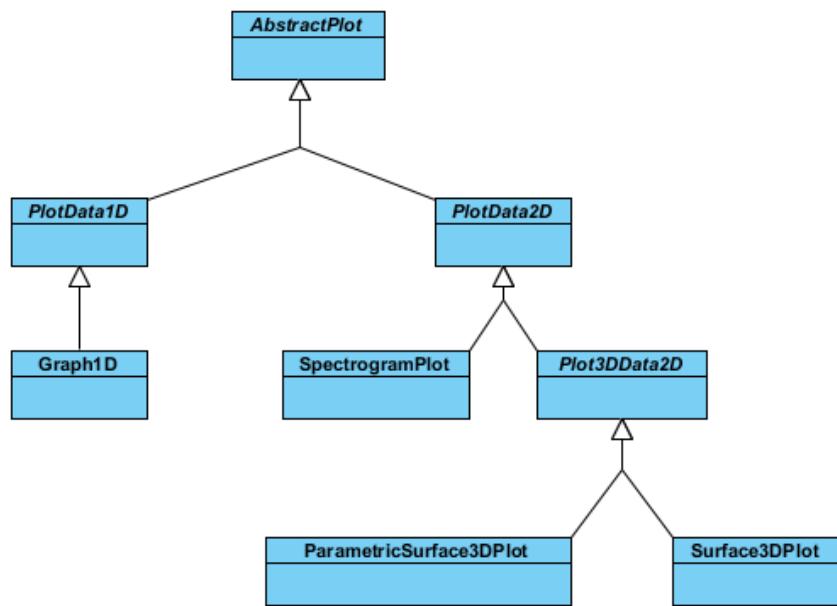


Figura 64. Diagrama con las clases gráficas

Los cuatro tipos de gráficos son los siguientes:

- **Graph1D**: Implementa un gráfico tipo “eje X/eje Y” para representar un objeto del tipo Data1D, obteniendo un resultado como el que se puede observar sobre la Figura 65. Sobre este gráfico se pueden realizar operaciones de zoom a tiempo real, así como desplazamientos del gráfico.
- **SpectrogramPlot**: Implementa un espectrograma para representar un objeto tipo Data2D, donde los colores representarán la variable dependiente, y los ejes X-Y las variables independientes, obteniendo un resultado como el que se puede observar en la Figura 66. Sobre este gráfico también se pueden realizar

operaciones de zoom a tiempo real, así como desplazamientos del gráfico, además de poder activar y desactivar las curvas de nivel.

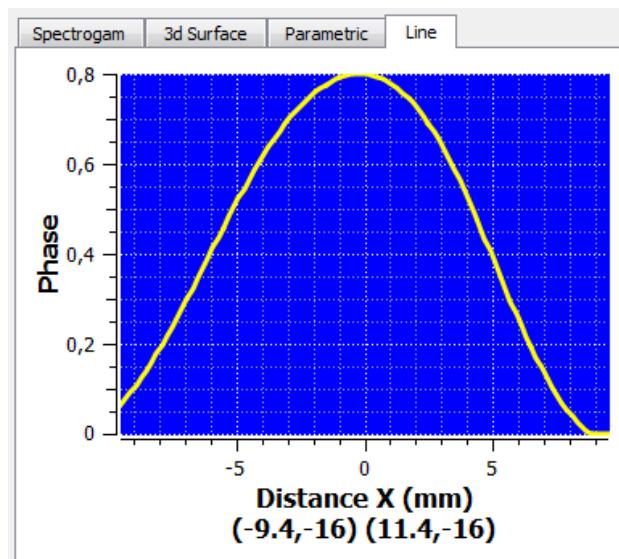


Figura 65. Imagen del gráfico Graph1D

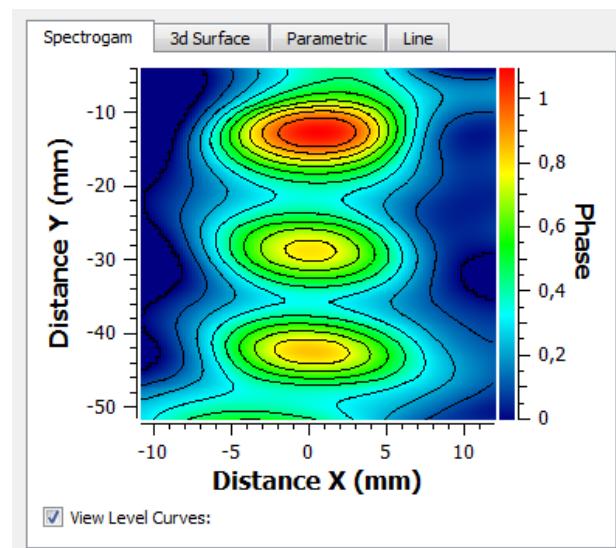


Figura 66. Imagen del gráfico SpectrogramPlot

- **Surface3DPlot:** Representa un objeto tipo Data2D como una superficie en un espacio de 3 dimensiones, siendo el eje Z la variable dependiente y los ejes X e Y las variables independientes, obteniendo un resultado como el que se puede observar en la Figura 67. Este gráfico se puede acercar y alejar, así como rotar y mover con el ratón en todas las direcciones a tiempo real.

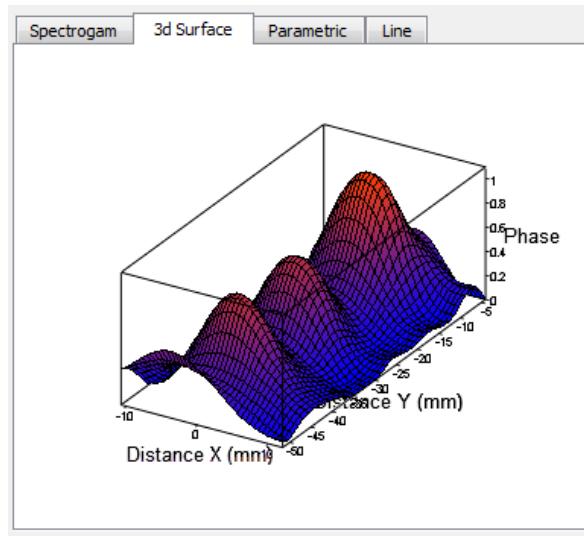


Figura 67. Imagen del gráfico Surface3DPlot

- **ParametricSurface3DPlot:** Esta clase está diseñada para reconstruir la imagen del jet de gas a partir de un objeto Data2D, mostrando la superficie donde la variable independiente alcanza cierto valor límite o *threshold*. Para que este gráfico tenga significado, es fundamental que el objeto que se esté midiendo en el sistema tenga una simetría cilíndrica como es el caso del jet de gas. El resultado obtenido se puede observar en la Figura 68. Este gráfico también se puede acercar y alejar, así como rotar y mover con el ratón en todas las direcciones a tiempo real.

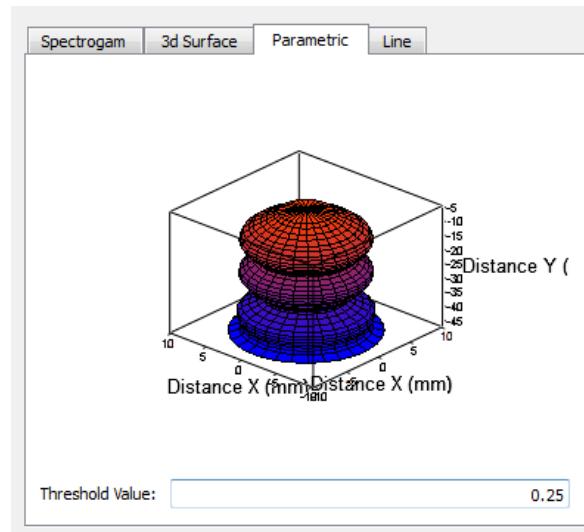


Figura 68. Imagen del gráfico ParametricSurface3DPlot

Nótese que, en general, la mayor dificultad a la hora de representar los gráficos, ha estado en crear clases que permitieran representar secuencias de varias imágenes por segundo evitando parpadeos, fugas de memoria, gestionando bien los rangos de los ejes y permitiendo mientras tanto al usuario realizar operaciones sobre el gráfico que se mantuvieran durante la transición de imágenes.

### 5.3.2. Ventanas y diálogos disponibles

Existen una serie de ventanas y cuadros de diálogo de usuario disponibles para el manejo de la aplicación. En este apartado nos limitaremos a mostrarlas y resumir brevemente su funcionalidad. En el Anexo III, de diseño, se pueden encontrar todos los detalles acerca de las clases relacionadas con cada una de ellas, y en el Anexo V, todos los manuales de uso.

La ventana principal es el elemento central de toda la aplicación y tiene una apariencia como la que se puede apreciar en la Figura 69.

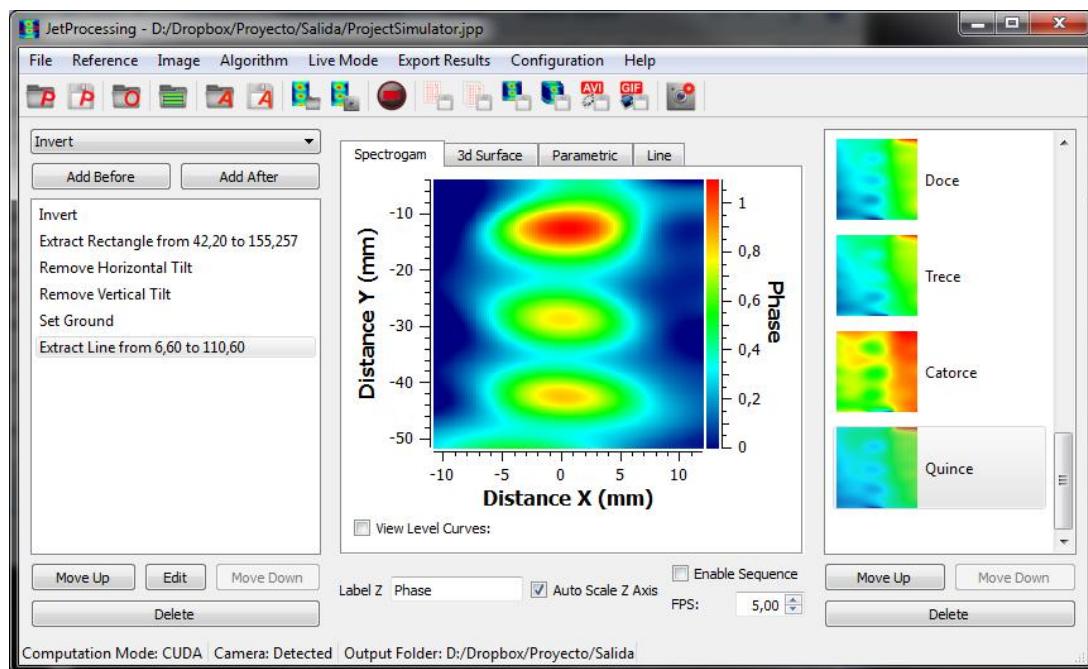


Figura 69. Imagen de la ventana principal

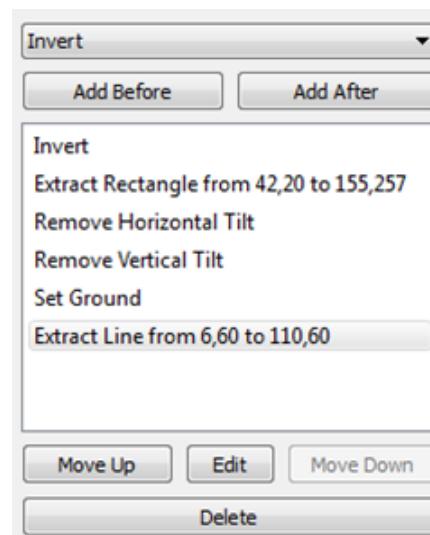
Es la ventana que permite el acceso a toda la funcionalidad del programa. Los elementos a destacar son los siguientes:

- Una **barra de título** mostrando el nombre de la aplicación, la ruta del proyecto actualmente abierto y, un asterisco, en caso de que el proyecto actual no esté guardado.
- Un **menú** para acceder a todas las funciones del programa.
- Una **barra de herramientas** con iconos creados especialmente para la aplicación, con acceso a las funciones más habituales.



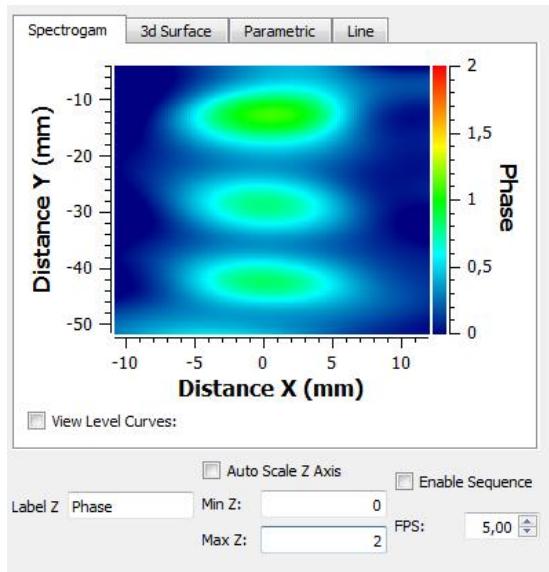
*Figura 70. Barra de herramientas*

- Una **lista de algoritmos** junto con todos los controles necesarios para añadir, mover, editar o eliminar los algoritmos de la lista. De cada algoritmo se muestra el nombre y los parámetros asociados.



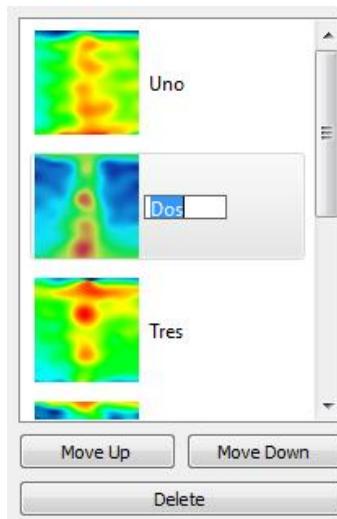
*Figura 71. Lista de algoritmos*

- Una **zona central de gráficos** junto con unos controles que permiten cambiar la etiqueta de la variable dependiente, escalar de forma manual o auto-escalar de forma automática la variable dependiente, y activar la vista en secuencia de las imágenes, seleccionando la velocidad de paso.



*Figura 72. Zona central de gráficos*

- Una **lista de imágenes**, incluyendo una etiqueta y una vista previa de cada una de ellas, con los correspondientes controles. Nótese que es posible cambiar la etiqueta de una imagen pinchando sobre ella dos veces o pulsando F2.



*Figura 73. Lista de imágenes*

- Una **barra de estado** que muestra información acerca del estado actual de la cámara, el modo de cálculo y la ruta de salida predeterminada.

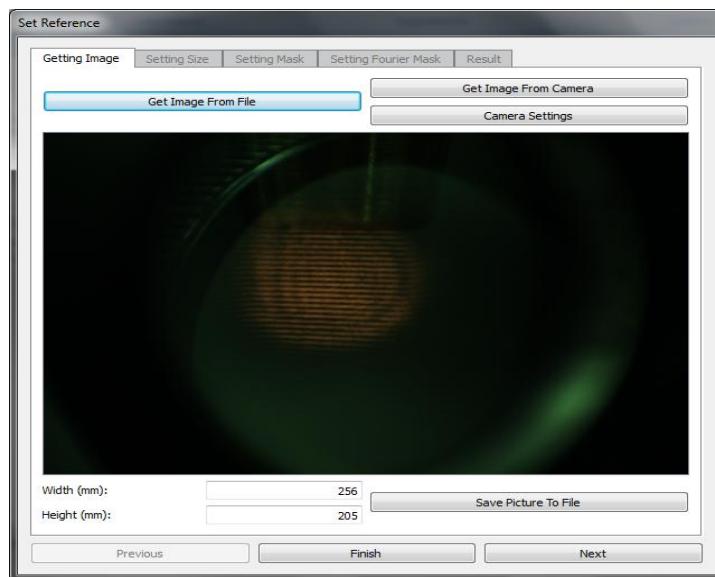
Computation Mode: CUDA Camera: Detected Output Folder: D:/Dropbox/Proyecto/Salida

*Figura 74. Barra de estado*

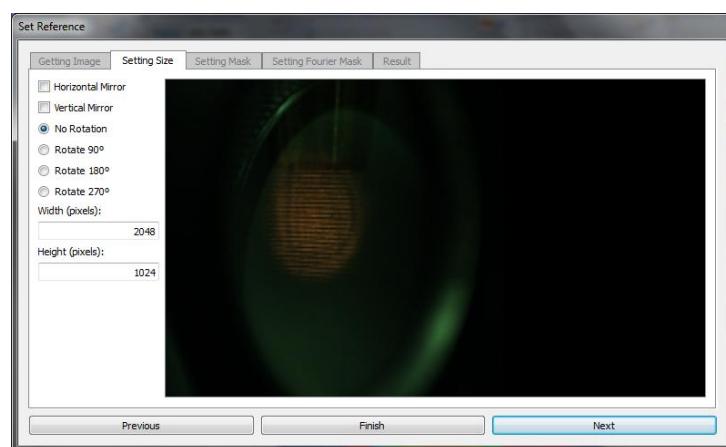
El siguiente cuadro de diálogo, por importancia, es el que permite fijar la imagen de referencia. Este cuadro de diálogo consta de varias pestañas, que van acompañando al usuario en el proceso de selección de la imagen de referencia y de los parámetros de procesado.

Las fases de este proceso son:

- **Selección de la imagen:** En este paso, se selecciona el archivo de imagen o bien se toma una imagen con la cámara, pudiendo acceder a la configuración de ésta. Además se indican las dimensiones reales de la imagen. Figura 75.



*Figura 75. Selección de la imagen de la referencia*



*Figura 76. Fijando tamaño*

- **Fijando tamaño:** En este paso, se permiten realizar operaciones de rotación o espejo sobre la imagen de la referencia, pudiendo ver los cambios en una vista previa. Además se debe indicar el tamaño efectivo deseado que se usará para los cálculos con la imagen. Figura 76.
- **Selección de máscara y origen de coordenadas:** En este paso se permite seleccionar la zona de interés de la imagen, así como el punto que se tomará como origen de coordenadas. Estos parámetros se pueden seleccionar tanto con el ratón sobre la imagen de la izquierda, sobre la que se puede hacer zoom y desplazar con el ratón para mayor comodidad, como con los controles disponibles, y pudiéndose además importar/exportar a un archivo. La vista previa de la derecha permite apreciar la zona seleccionada con más detalle.

Figura 77.

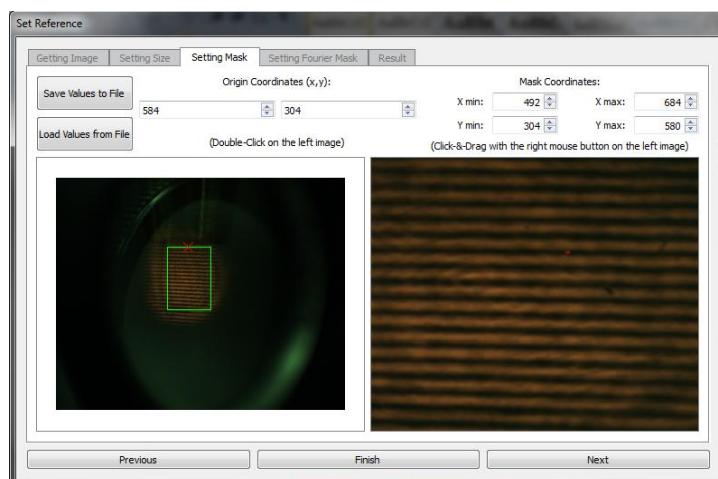


Figura 77. Fijando la máscara y el origen de coordenadas

- **Selección de la máscara de Fourier:** Este paso permite seleccionar una máscara para la transformada de Fourier de forma análoga al paso anterior. Además incluye un control para aumentar la ganancia de la vista previa para que el usuario pueda mejorar la visión de la imagen. Figura 78.
- **Vista del resultado de la fase:** En el último paso se muestra una vista previa de la fase de la referencia sin escanear, para que el usuario pueda comprobar si el resultado es bueno, o por el contrario debe modificar algunos de los parámetros. Figura 79.

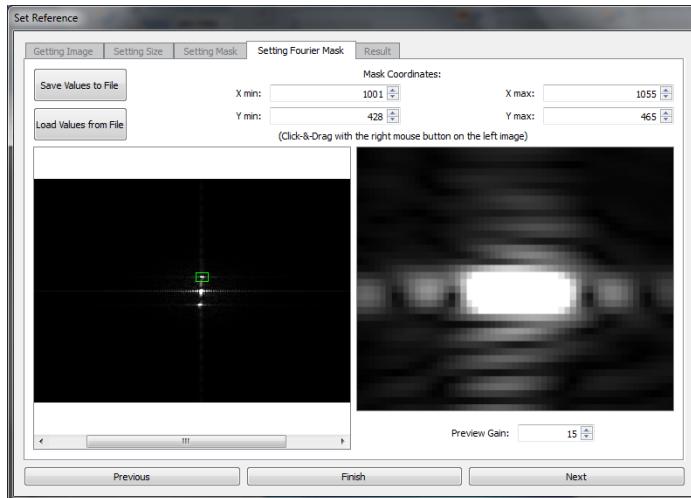


Figura 78. Fijando la máscara de Fourier

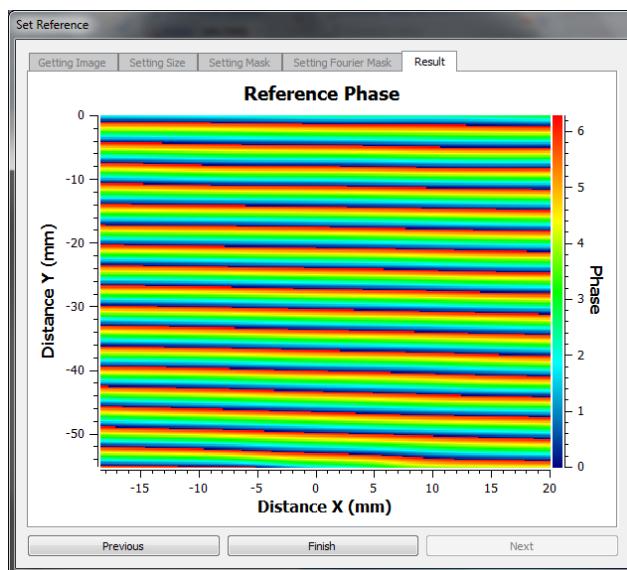


Figura 79. Resultado para la fase de la referencia

Dado que algunos algoritmos llevan parámetros, existe un cuadro de diálogo para cada uno de ellos, que se lanza cada vez que se añade o edita un algoritmo de ese tipo.

El cuadro de diálogo más completo es aquel que permite seleccionar una línea del gráfico, como el que se puede contemplar en la Figura 80. En él se puede seleccionar una recta, moverla a gusto del usuario pulsando en el centro o en sus extremos, y además, observar durante todo el tiempo una vista previa a la derecha con el gráfico de la línea seleccionada.

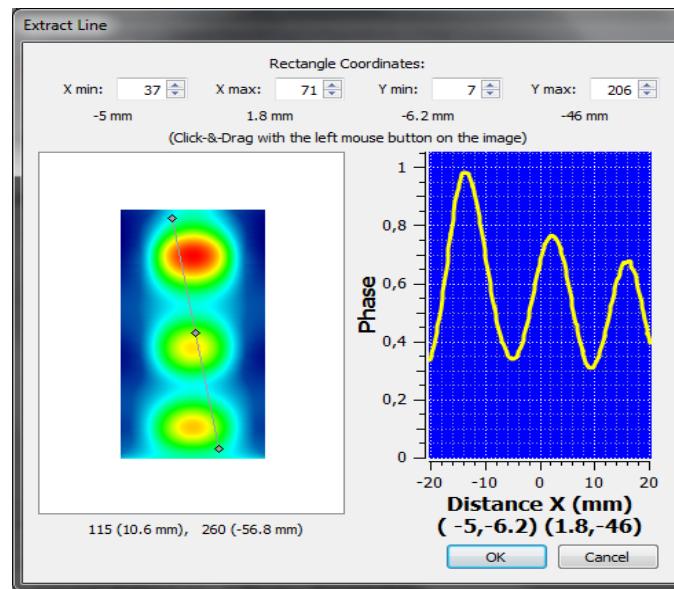


Figura 80. Diálogo de selección de recta

Otros cuadros de diálogo de introducción de parámetros son los siguientes:

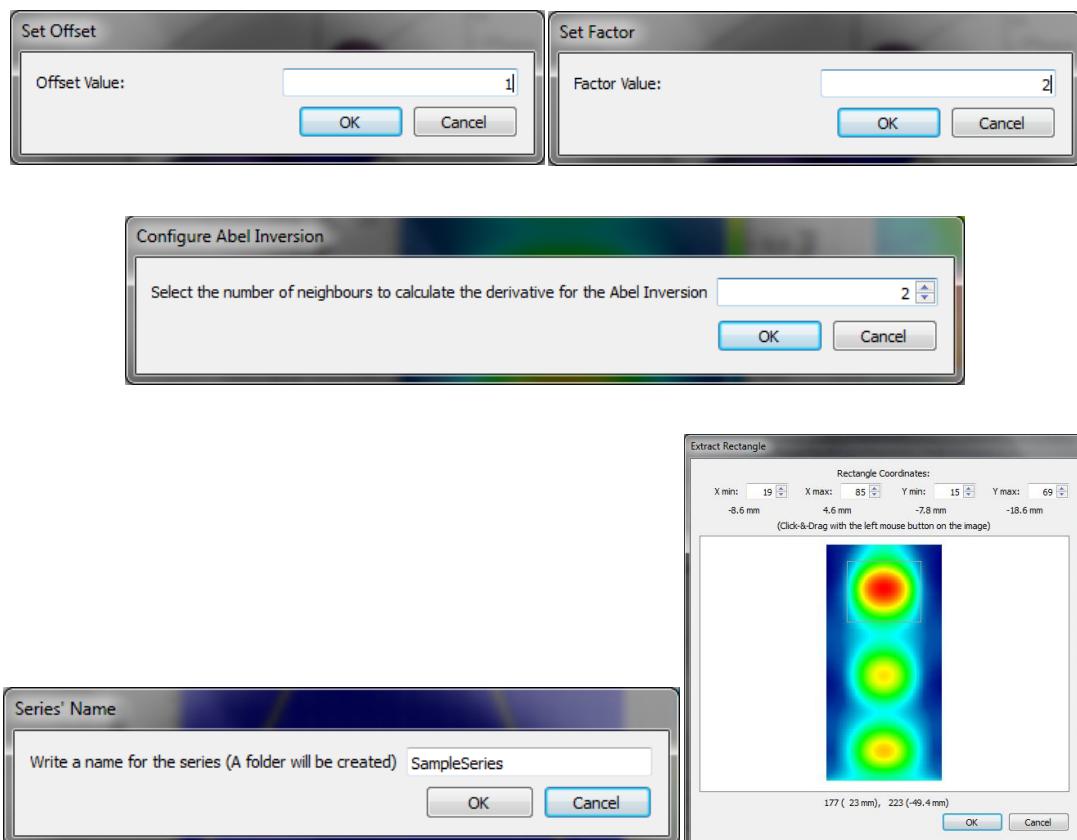
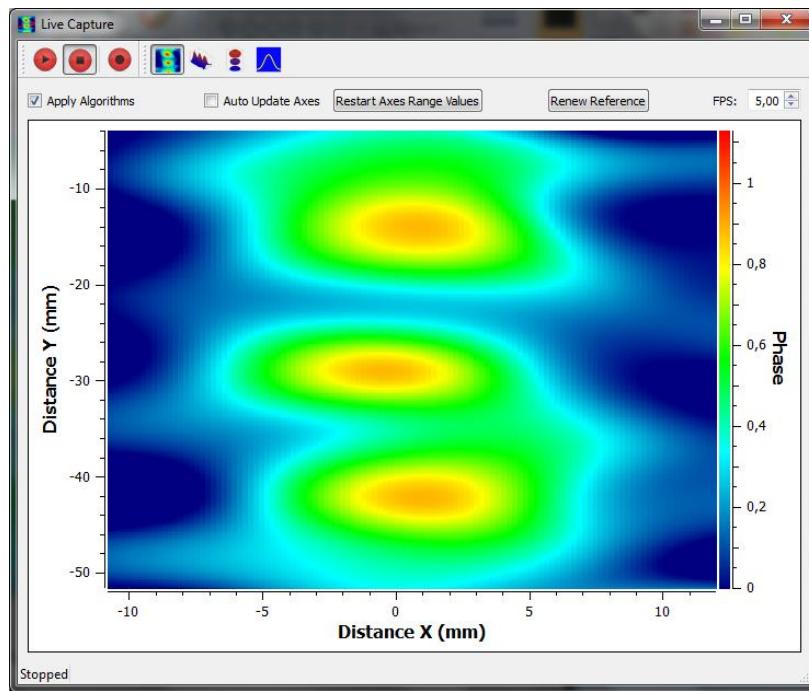


Figura 81. Otros cuadros de diálogos de introducción de parámetros

La siguiente ventana que se va a presentar es la que corresponde al modo “en vivo”, y se puede ver en la Figura 82.



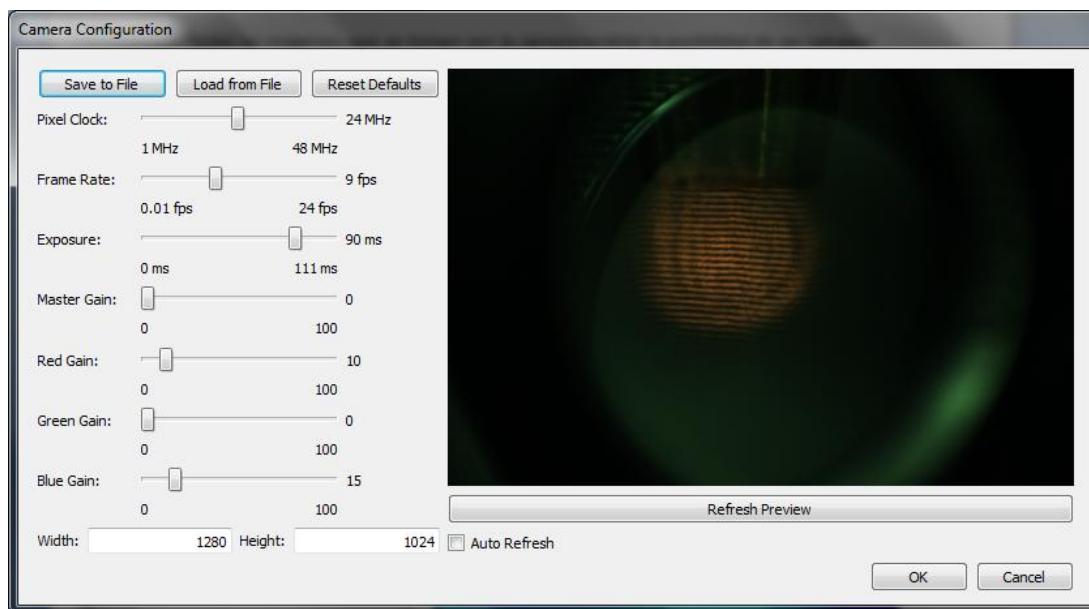
*Figura 82. Ventana del modo “en vivo”*

Esta ventana es la que permite visualizar el jet de gas a tiempo real, controlando el proceso mediante los siguientes elementos:

- Unos botones de navegación: que permiten comenzar y parar el proceso de visualización, así como almacenar las imágenes tomadas en la lista de la interfaz principal.
- Unos botones de selección de gráfico , y su gráfico asociado, permitiendo mostrar los mismos tipos de gráficos que en la ventana principal para la imagen en tiempo real.
- Un control de velocidad **FPS: 5,00**: que permite fijar la tasa deseada de imágenes por segundo.
- Una casilla  **Apply Algorithms**: que permite activar o desactivar la aplicación de la lista de algoritmos a la imagen que se está mostrando.

- Una casilla  **Auto Update Axes** : que permite activar la actualización automática de los rangos de los ejes con cada imagen. Si se desactiva, se irá siempre ampliando el rango de los ejes en base a las siguientes imágenes que se vayan procesando.
- Un botón **Restart Axes Range Values** : que permite reiniciar los rangos de los ejes en el caso de que el auto escalado de los ejes esté desactivado y se haya recibido una imagen con ruido provocando la selección de un rango demasiado grande, especialmente para el eje z.
- Un botón **Renew Reference** : que permite renovar la imagen de la referencia con una imagen tomada por la cámara en el momento en que se pulsa, manteniendo iguales los parámetros de procesado.
- Una barra de estado **Playing at 5.15 fps**: indicando el estado del proceso y el valor efectivo de imágenes procesadas por segundo.

Y por último, quedan los cuadros de diálogo relacionados con configuración:



*Figura 83. Cuadro de diálogo de configuración de la cámara*

- El **cuadro de diálogo de configuración de la cámara** permite modificar los parámetros de la cámara, cargarlos y salvarlos a un archivo o establecerlos a los

valores por defecto. Además muestra una vista previa (Periódicamente, o cuando el usuario lo solicite) de una imagen tomada con los parámetros actuales. Además, se modifican los valores máximos y mínimos de los parámetros en función de los demás parámetros. Figura 83.

- El **cuadro de diálogo de selección de modo** que permite elegir entre los dos modos de cálculo disponibles: CUDA, o la CPU. Se mostrará siempre que al iniciar el programa la aplicación detecte que CUDA está disponible. Es posible deshabilitar este cuadro de diálogo para las posteriores ejecuciones mediante el uso de una casilla incluida para ello. Figura 84.

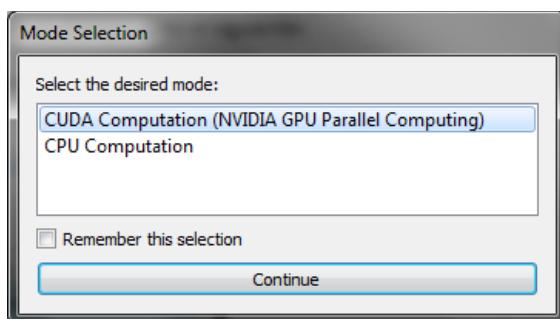


Figura 84. Diálogo de selección de modo

- El **cuadro de diálogo de preferencias del programa** (Figura 85) permite modificar algunas opciones, como la aparición del cuadro de selección de modo de cálculo, el formato y tamaño por defecto de las imágenes exportadas de los gráficos...

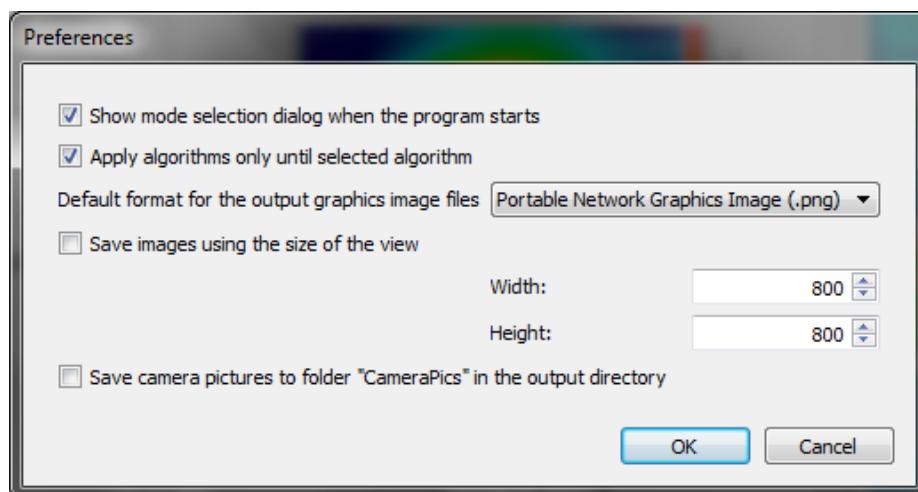


Figura 85. Diálogo de preferencias del programa

## 5.4. Algunos diagramas de clases de la vista estática

Una buena forma de hacerse una idea de la organización de una aplicación son los diagramas de clases. Dado que el programa consta de más de 60 clases distintas (más de 70 incluyendo las clases generadas por Qt) se ha generado un gran número de diagramas que pueden ser consultados con detalle en el Anexo III de Diseño, así como en el Anexo IV de Manual del Programador.

El objetivo de este apartado es mostrar de una forma muy breve algunos de los diagramas de clases de la aplicación que se consideran de especial interés.

### 5.4.1. Estructura interna del programa

En la aplicación creada la interfaz gráfica está claramente separada del control y la lógica del programa. En este apartado, se muestran los diagramas relacionados con la parte interna del programa, la que tiene el control de la funcionalidad y de la lógica de la aplicación, así como acceso a los datos, sin incluir ninguna clase relacionada con la interfaz.

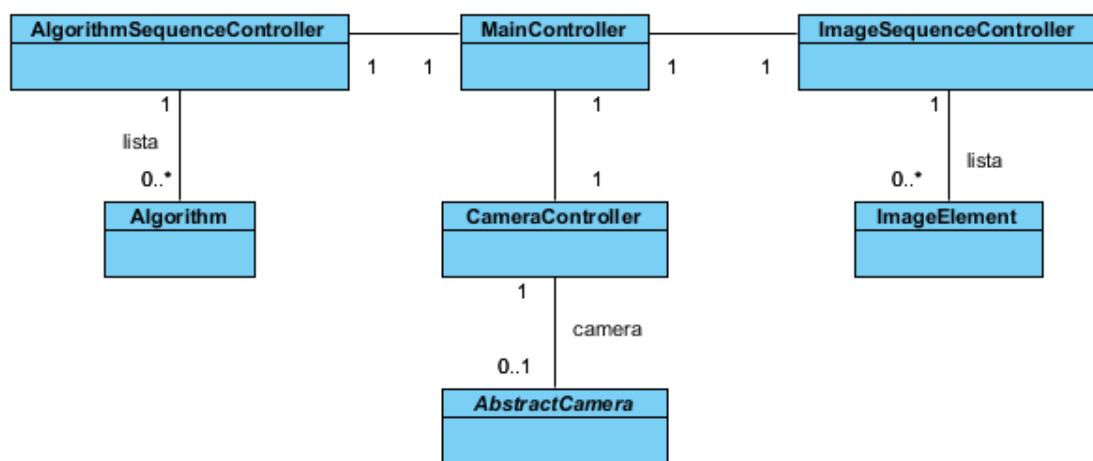


Figura 86. Estructura interna del programa a primer nivel

Un primer nivel de la relación de las clases internas se puede observar en la Figura 86. El programa consta de un controlador principal, que delega responsabilidad en otros tres controladores, el controlador de algoritmos, el controlador de imágenes y el controlador de la cámara.

El objetivo, es que cada controlador secundario sea independiente realizando las tareas específicas sobre la lista de algoritmos, sobre la lista de imágenes, o sobre la cámara, mientras que el controlador principal coordine todo el proceso que relate a los otros controladores entre sí, así como implemente la funcionalidad general del programa.

#### 5.4.2. Diagramas de clases con las estructuras y parámetros de cálculo

En los diagramas que se representan a continuación, se muestran las clases relacionadas con las estructuras y parámetros necesarios para el cálculo.

El procesado completo de una simple imagen tomada con la cámara, implica el acceso a los tres controladores secundarios, por lo que será una tarea del controlador principal.

El controlador principal está rodeado, entre otras, de todas las clases relacionadas con la imagen de referencia y los parámetros para el pre-procesado, siguiendo un esquema como el que se puede observar en la Figura 87.

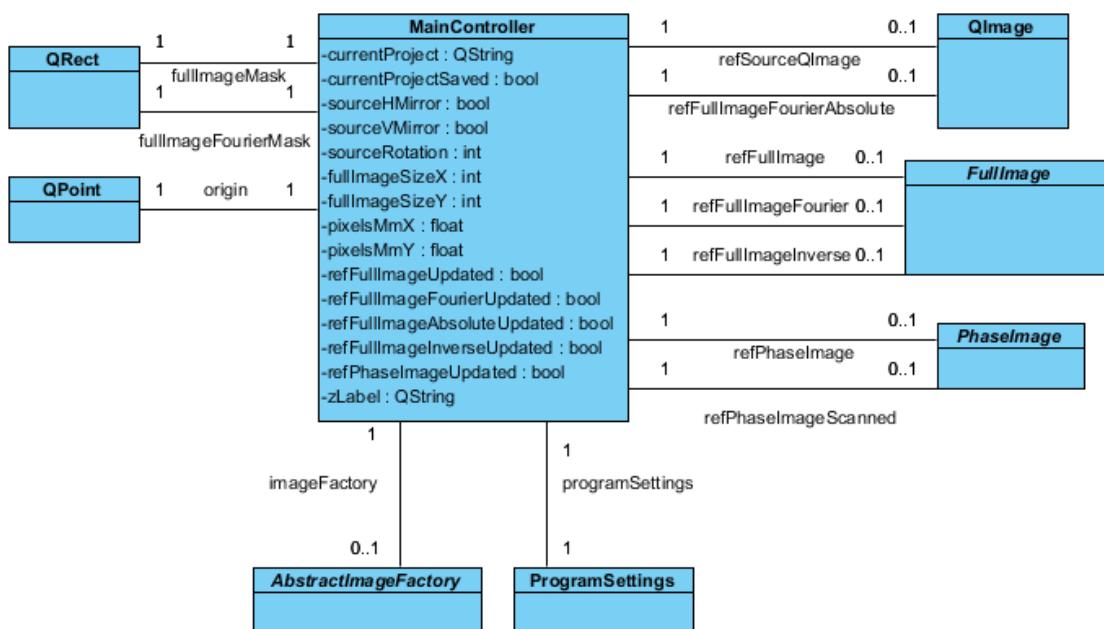
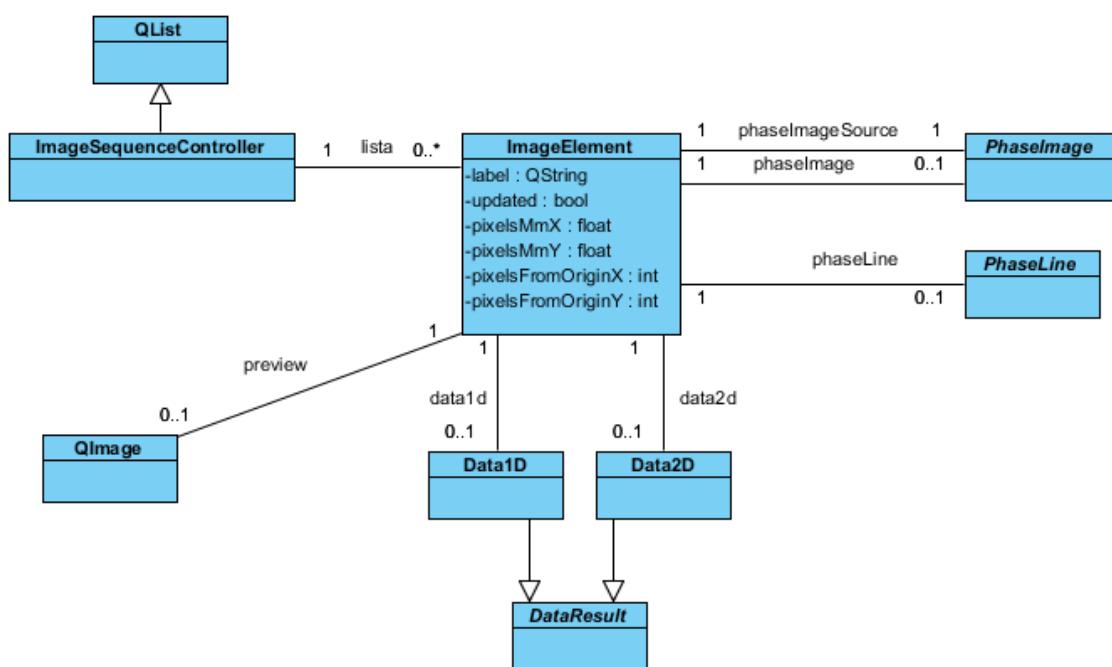


Figura 87. Controlador principal y sus relaciones a segundo nivel

Por otra parte, el controlador de la cámara y el controlador de algoritmos están relacionados con la cámara y la lista de algoritmos, respectivamente, para tomar la imagen, y conocer los algoritmos con sus parámetros asociados que se deben utilizar para el posterior procesado de la primera imagen obtenida del jet.

Y en último lugar, como se puede observar en la Figura 88, el controlador de imágenes tiene relaciones para poder almacenar la lista de imágenes guardando para cada una de ellas la imagen (Tanto antes, como después de ser procesada), los resultados generados e incluso una vista previa.



*Figura 88. Controlador de imágenes y sus relaciones a segundo nivel*

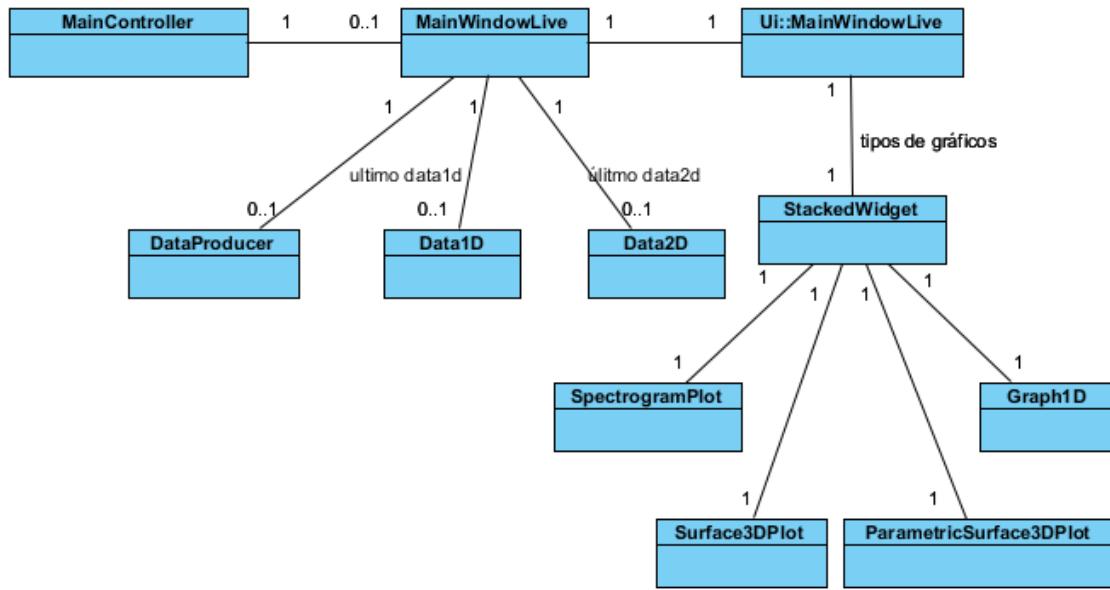
#### 5.4.3. Ejemplo de diagrama de clases para una ventana

Como ya se comentó, todas las ventanas o diálogos, constan de una clase para la interfaz gráfica, generada automáticamente por Qt, que comienza por “Ui::”, una clase controladora específica para cada diálogo, y acceso a la lógica del programa, a través del controlador principal, u otro controlador.

El controlador de la interfaz será quien se encargue de implementar la funcionalidad directamente relacionada con dicha interfaz, como puede ser, una selección con un

rectángulo que se mueve con el ratón, la activación o desactivación de propiedades en los gráficos, responder a señales que han generado los elementos de la interfaz...

Un ejemplo sencillo de este tipo de organización es el que se puede observar en el diagrama de clases relacionado con la ventana del modo “en vivo”, que se puede observar en la Figura 89.



*Figura 89. Diagrama de clases de la ventana del modo “en vivo”*

La clase **Ui::MainWindowLive**, implementa la interfaz gráfica, con los distintos gráficos, y demás elementos, mientras que la clase **MainWindowLive**, se encarga de recibir las señales que **Ui::MainWindowLive** genera y de representar cuando corresponda los datos tipo **Data1D** y **Data2D** a los que tiene acceso, que le ha ido pidiendo al controlador principal, a través de la relación que hay entre ellos.

### 5.5. Conviviendo CUDA y CPU: Factoría Abstracta

Uno de los objetivos/requisitos del proyecto era la programación de los cálculos con la herramienta CUDA de NVIDIA, consiguiendo con ello una mayor velocidad.

Sin embargo, no se quiso dejar fuera la posibilidad del trabajo simplemente con la CPU para los ordenadores que no dispusieran de una tarjeta compatible con CUDA, ya que

el requisito de velocidad es sólo estrictamente necesario para el modo “en vivo” de toma y procesado de imágenes, pero no para el procesado de imágenes individuales, o en grupo directamente desde archivos.

Esto llevó a realizar dos implementaciones para todos los cálculos: con la CPU, y con CUDA, entre las que se debe elegir, de forma automática o manual al comenzar la ejecución del programa.

No obstante, esta doble implementación no es inmediata de gestionar, ya que si no se hace bien, se puede acabar con todo el código del programa lleno de condicionales dependiendo del tipo de cálculo que se esté realizando.

Para evitar esto, en JetProcessing, se ha recurrido al uso de abstracción con herencia y factorías (*Patrones Abstract Factory y Factory Method*), siguiendo el esquema que se puede observar en la Figura 90.

La primera parte de la solución gira en torno a la abstracción de las clases de cálculo.

Existen 3 tipos de clases abstractas que representan una entidad sobre la que se realizan cálculos:

- **FullImage:** representa la imagen en las etapas del pre-procesado hasta obtener la fase. Contiene métodos abstractos para realizar todas las operaciones necesarias en esa etapa, así como para obtener una imagen a partir de ella, o un objeto del tipo **PhaselImage**.
- **PhaselImage:** representa una imagen de la fase sobre la que se pueden aplicar los algoritmos de procesado. Contiene métodos abstractos para aplicar esos algoritmos, obtener un objeto del tipo **PhaseLine** o devolver un objeto tipo **Data2D** con los datos de la imagen para su representación o exportado.
- **PhaseLine:** representa la fase a lo largo de una recta sobre la que se pueden aplicar algoritmos de procesado. Contiene también por lo tanto métodos abstractos para aplicar esos algoritmos, así como para obtener un objeto tipo **Data1D** con los datos de la línea para su representación o exportado.

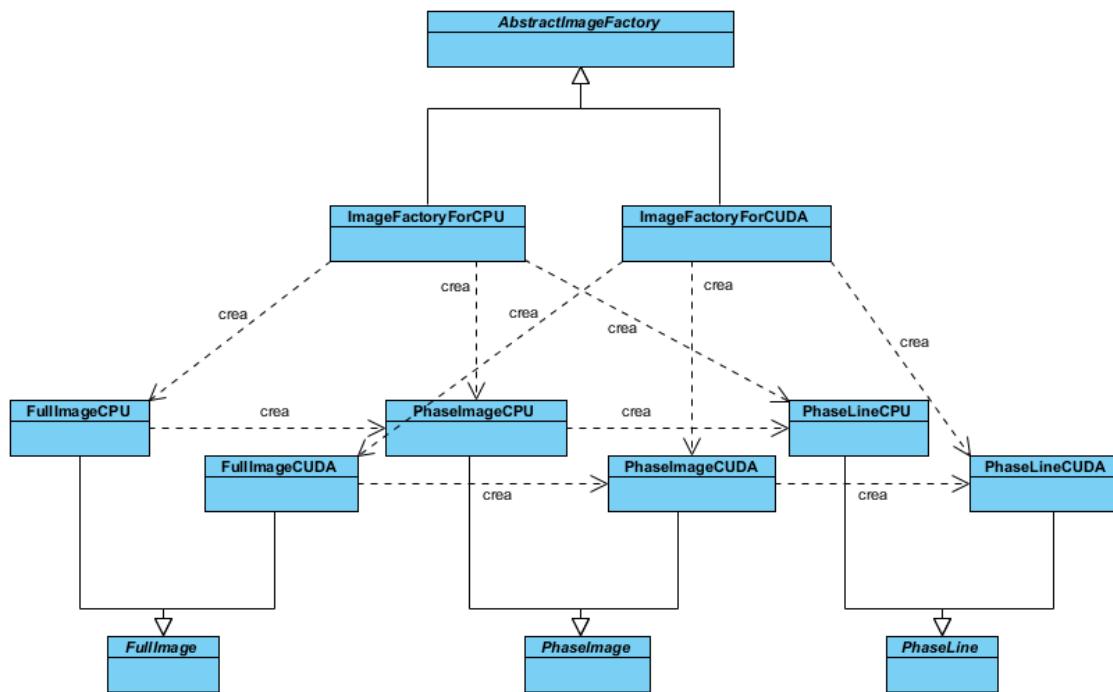


Figura 90. Factorías para los cálculos

De cada una de esas clases abstractas, derivan otras dos clases, con el mismo nombre, añadiendo al final CUDA o CPU. Cada una de estas clases hija, implementa todos los métodos de la clase padre, de acuerdo al método de cálculo que le corresponde, y creando cuando sea necesario, objetos de su misma implementación, es decir, un objeto tipo “FullImageCUDA”, con una llamada al método “*PhaselImage \* CropPhase(QRect mask);*” creará un objeto del tipo “*PhaselImageCUDA*”.

Con esto, habríamos reducido el problema a la creación y copia de los objetos: con tener un puntero a un objeto tipo *PhaselImage*, y llamar a sus métodos, el código es independiente del tipo de implementación del objeto al que apunte el puntero, excepto en el momento de la creación, como se puede ver en el siguiente ejemplo:

```

//Creo el objeto con la implementación que corresponda
FullImage * fullImage;
if(mode==MODE_CUDA)
    fullImage= new FullImageCUDA
        (&auxImage,fullImageSizeX,fullImageSizeY);
else if(mode==MODE_CPU)
    fullImage= new FullImageCPU
        (&auxImage,fullImageSizeX,fullImageSizeY);
.
.
.
  
```

```
//Aplico operaciones con la implementación que elegí, sin que
//el código dependa de la implementación
fullImage->ApplyMask(fullImageMask);
fullImage->ComplexFFT(FFT_FORWARD);
fullImage->ReArrangeFT();
fullImage->ApplyMask(fullImageFourierMask);
fullImage->ReArrangeFT();
fullImage->ComplexFFT(FFT_INVERSE);
.
.
.

//E incluso obtengo otro objeto y elimino el primero, sin
//dependencias con la implementación
PhaseImage * phaseImage;
phaseImage=fullImage->CropPhase(fullImageMask);
delete fullImage;
```

No obstante, aunque ya se tiene un código mejor organizado, sigue estando el problema de la copia y la creación, ya que cada vez que se creara/copiera un objeto, habría que recurrir al condicional.

Para solucionar eso, se recurre al uso de factorías, abstractas y concretas. La idea, es que sea la factoría la encargada de crear/copiar los objetos, y devolvernos un puntero a ellos.

En este caso tenemos tres clases implicadas:

- **AbstractImageFactory**: Es una clase abstracta que tiene métodos también abstractos para la creación y copia de objetos del tipo FullImage, PhaseImage y PhaseLine, recibiendo los mismos argumentos que reciben los constructores de estos.
- **ImageFactoryForCPU**: Es una clase que deriva de AbstractImageFactory y que implementa sus métodos, devolviendo objetos del tipo FullImageCPU, PhaseImageCPU y PhaseLineCPU.
- **ImageFactoryForCUDA**: Es una clase que deriva de AbstractImageFactory y que implementa sus métodos, devolviendo objetos del tipo FullImageCUDA, PhaseImageCUDA y PhaseLineCUDA.

Y la forma de trabajar, es la siguiente:

- Al comenzar la ejecución del programa, se crea una factoría concreta, en función del tipo de cálculo que se quiera:

```
AbstractImageFactory * imageFactory;
switch(programSettings.GetMode()) {
    case MODE_CUDA:
        imageFactory = new ImageFactoryForCUDA(); break;
    default:
        imageFactory = new ImageFactoryForCPU();
}
```

- Después, cada vez que se quiera crear/copiar uno de los objetos con doble implementación, se usará la factoría:

```
//Creando objetos
FullImage * fullImage;
fullImage = imageFactory->CreateFullImage
    (&auxImage, fullImageSizeX, fullImageSizeY);
.

.

.

//Aplicando operaciones
fullImage->ApplyMask(fullImageMask);
fullImage->ComplexFFT(FFT_FORWARD);
fullImage->ReArrangeFT();
fullImage->ApplyMask(fullImageFourierMask);
fullImage->ReArrangeFT();
fullImage->ComplexFFT(FFT_INVERSE);
.

.

.

//Copiando objetos
FullImage * fullImageCopy;
fullImageCopy = imageFactory->CopyFullImage(fullImage);
.

.

.

//Obteniendo otros objetos
PhaseImage * phaseImage;
phaseImage=fullImage->CropPhase(fullImageMask);
.

.

.

//Eliminando objetos
delete fullImage;
```

De esta forma se asegura que en todo el programa se están usando objetos con la misma implementación, y, como se puede observar en los ejemplos, solamente hay condicionales de la implementación en la creación de la factoría al principio de la ejecución del programa, quedando un código totalmente limpio e independiente del tipo de cálculo.

Para ver cómo los objetos utilizan la factoría, consúltese el capítulo 5.8.

### ***5.6. Optimización de código para obtener rendimiento***

Uno de los aspectos más delicados a la hora del diseño del proyecto, ha consistido en buscar un compromiso entre una programación elegante, con un buen estilo, reutilizable y poco acoplada, y una programación eficiente, para obtener el rendimiento que se buscaba.

Como ya se indicó, uno de los motivos para elegir el lenguaje C++ fue la posibilidad de trabajar con punteros, para optimizar la gestión de memoria. Sin embargo, un puntero a una zona de memoria, es una característica de programación de muy bajo nivel, ya que quien usa el puntero tiene que saber “qué hay y cómo de grande” es la zona de memoria a la que está apuntando, creando un código muy acoplado.

La forma de gestionar eso, es la creación de un objeto, que sepa todo lo que tenga que saber de la información a la que apunta el puntero, y encapsule toda esa dificultad, pero que de la misma forma sea eficiente haciendo las menores copias de memoria posibles, y sobre todo, el menor número de transferencias entre memoria principal y memoria de la GPU posibles.

Para ello, se crearon los objetos indicados en el apartado anterior: FullImage, PhaselImage y PhaseLine, conteniendo cada uno de ellos un puntero a una zona de memoria.

Estos punteros podrán apuntar o bien a memoria principal, en el caso de implementación con la CPU, o bien a memoria de la tarjeta gráfica, en caso de implementación con CUDA.

De esta forma una implementación con CUDA tendrá los datos permanentemente en la memoria de la GPU, siendo la secuencia de eventos la siguiente:

1. Se crea un objeto con tipo FullImageCUDA, a partir de una imagen tipo QImage, cuyo contenido se transfiere de memoria principal a memoria de la GPU.

2. Se utilizan métodos del objeto tipo FullImageCUDA, evitando transvases de memoria, siempre trabajando sobre la memoria de la GPU.
3. Con un método de FullImageCUDA, se extrae la fase de cierta zona a un objeto del tipo PhaselImageCUDA, todo ello sin tener que transferir ninguna zona de memoria, con el nuevo objeto alojado directamente en la GPU.
4. Se utilizan métodos del objeto tipo PhaselImageCUDA para procesar la imagen, también evitando transvases de memoria, siempre trabajando sobre la memoria de la GPU.
5. Cuando se necesitan los datos de la imagen, se llama al método Data2D \* GetData2D() del objeto tipo PhaselImageCUDA, que crea un objeto tipo Data2D, haciendo un transvase desde la memoria de la GPU a la memoria principal, que será independiente de la implementación y apto para la representación o exportación.

Trabajando así se ha conseguido, por una parte, tener una buena estructura de objetos con mucho encapsulamiento, que por otra parte, realizan las operaciones de una forma óptima, ya que solamente ha habido dos transferencias inevitables de memoria entre la memoria principal y la memoria de la GPU: al principio y al final de todo el procesado.

Nótese que si se hubiera extraído una línea de fase, se habrían añadido dos pasos entre los pasos 4 y 5: crear un objeto tipo PhaseLineCUDA con un método del objeto tipo PhaselImageCUDA, y aplicar operaciones sobre ese objeto, obteniendo en el último paso en este caso un objeto del tipo Data1D, manteniendo solamente dos transvases de memoria entre la CPU y la GPU en total.

### **5.7. Paralelización de algoritmos para CUDA**

En esta sección se resume brevemente el proceso de paralelización en CUDA para algunas de las operaciones que se realizan sobre las imágenes, indicando las fases que se siguen y el número de bloques e hilos máximo que se podrían llegar a tener en paralelo.

Se omiten aquellas que se limitan a aplicar una operación punto a punto, por ser muy parecidas a las del ejemplo visto en el apartado 4.6.3.

Nótese que para la implementación con la CPU, simplemente se realizan las mismas operaciones de forma secuencial.

### 5.7.1. Escaneo de la fase

Para realizarlo se utiliza una matriz auxiliar de enteros del mismo tamaño que la imagen de la fase que se quiere escanear. En cada cuadro de la matriz, se almacenará el número de saltos de fase que se han encontrado para llegar desde el centro de la imagen hasta el punto en cuestión. Nótese que estos saltos de fase pueden ser positivos o negativos en función si van de 0 a  $2\pi$ , o de  $2\pi$  a 0.

El escaneo de la fase se realiza en tres pasos:

1. Hay dos hilos en paralelo. Uno asigna el 0 al punto central de la imagen en la matriz auxiliar, y después, comienzan, uno de ellos, hacia la derecha, y otro de ellos hacia la izquierda, a comparar los valores de la fase e ir escribiendo en la matriz auxiliar los saltos de fase que han encontrado hasta llegar a cada punto, evaluando el salto punto a punto, y asignando a cada punto el valor del punto anterior, igual, incrementado, o disminuido, en función de la ausencia o tipo de salto encontrado. Cada hilo procesa “el ancho de la imagen dividido entre dos” puntos.

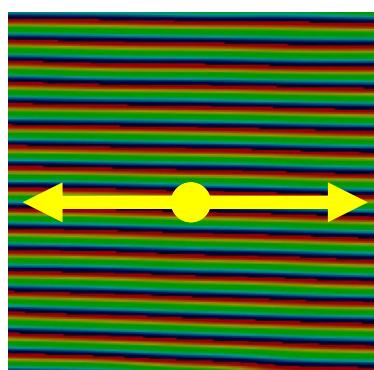
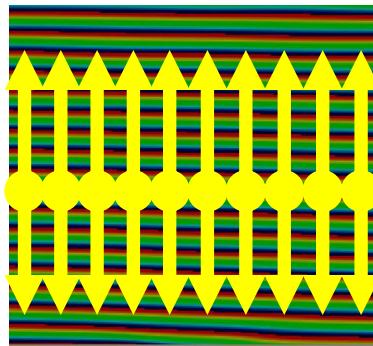


Figura 91. Escaneado de fase: Paso 1

2. Una vez fijados los saltos para la línea horizontal central, se lanzan el “ancho de la imagen por dos” hilos, distribuidos en bloques de 32, que se ejecutarán idealmente en paralelo. Cada hilo se encargará de partir de un punto de la recta horizontal central y recorrer la recta vertical asociada a dicho punto, o bien hacia arriba, o bien hacia abajo, rellenando la matriz auxiliar con los saltos de fase encontrados, de la misma forma que antes.



*Figura 92. Escaneado de fase: Paso 2*

3. Una vez rellena la matriz auxiliar, el tercer paso solamente consiste en añadir punto a punto a cada punto de la matriz el valor  $2\pi$  multiplicado por los saltos de fase almacenados en la matriz asociada en el lugar correspondiente. Esta operación por ser punto a punto se distribuye de igual forma que el ejemplo del apartado 4.6.3.

$$\text{imagen}[x][y] = \text{imagen}[x][y] + 2\pi \cdot \text{auxiliar}[x][y]$$

### 5.7.2. Eliminación de la inclinación horizontal/vertical

Para realizarlo, se utiliza un vector auxiliar de flotantes, del mismo tamaño que el alto/ancho de la imagen que se va a procesar, así como un flotante para el valor medio. En cada fila del vector se almacenará el valor de la inclinación encontrado en esa recta horizontal/vertical.

Se realiza en 3 pasos:

1. El “alto”/“ancho” de hilos, en bloques de 32, idealmente en paralelo. Cada uno calcula la pendiente de cada recta horizontal/vertical y lo almacena en la

posición adecuada del vector. Para ello cada hilo evalúa la siguiente expresión (Inclinación horizontal) para la recta que le corresponde:

$$m_i = \frac{\sum x_i \cdot \sum z_i - N \cdot \sum(x_i \cdot z_i)}{(\sum x_i)^2 - N \cdot \sum z_i^2}$$

2. Un solo hilo calcula la media aritmética de todos los valores de pendiente en el vector:  $\bar{m}$  será la inclinación que se eliminará.

$$\bar{m} = \frac{\sum_{i=1}^M m_i}{M}$$

3. Se elimina punto a punto la inclinación horizontal/vertical, aplicando la siguiente expresión (Inclinación horizontal):

$$z(x, y) = z(x, y) - \bar{m} \cdot (x - \frac{(x_{max} - x_{min})}{2})$$

Esta operación por ser punto a punto se distribuye de igual forma que el ejemplo del apartado 4.6.3.

### 5.7.3. Fijado del nivel base

Este algoritmo es muy sencillo, utiliza un flotante auxiliar y consta simplemente de dos pasos.

1. Un hilo en un bloque recorre todos los puntos de la recta vertical de más a la derecha y más a la izquierda, haciendo la media de todos los valores, para obtener el valor b.
2. Se aplica punto a punto la siguiente operación:

$$z(x, y) = \begin{cases} 0, & \text{si } z(x, y) < b \\ z(x, y) - b, & \text{si } z(x, y) \geq b \end{cases}$$

Por ser punto a punto se distribuye de igual forma que el ejemplo del apartado 4.6.3.

### 5.7.4. Simetrización

La simetrización utiliza un vector de enteros del tamaño del alto de la imagen en el que se almacena la posición del máximo, y una matriz igual que la imagen, copia de ella.

Se realiza en dos pasos:

1. El “alto” de hilos, en bloques de 32, en paralelo. Cada uno de ellos va recorriendo una línea horizontal, buscando la posición del máximo, que almacena en el vector de máximos.
2. Asignación punto a punto en la matriz de los datos simetrizados, a partir de los datos de la copia en la matriz auxiliar. Para ello, dado un punto cuyo valor se quiere calcular en una recta horizontal, se mira la distancia de ese punto al centro de la imagen, y se busca en la imagen original, los valores de los puntos que estaban a esa misma distancia, a la derecha y a la izquierda del máximo, calculando su media y asignándosela al punto. Esta operación por ser punto a punto se distribuye de igual forma que el ejemplo del apartado 4.6.3.

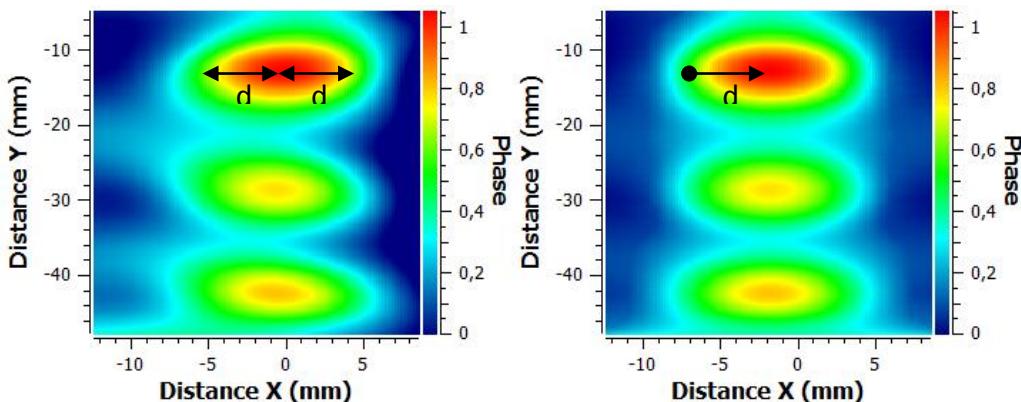


Figura 93. Simetrización: Paso 2

#### 5.7.5. Inversión de Abel

La inversión de Abel utiliza una matriz de flotantes auxiliar, del tamaño de la imagen de la fase, para guardar la derivada.

Se realiza en dos pasos:

1. En primer lugar se calcula la derivada horizontal. Para ello se compara el valor del punto en el que se quiere calcular la derivada, con los de los vecinos por la derecha y por la izquierda, en función del número de vecinos elegidos. Para cada vecino utilizado, se hace la diferencia con el valor del punto, y se divide por la distancia en píxeles. A continuación se hace la media de los valores, y se divide entre la distancia real entre píxeles para obtener las dimensiones

correctas. Esta operación por ser punto a punto se distribuye de igual forma que el ejemplo del apartado 4.6.3, pero escribiendo el resultado en la matriz auxiliar, ya que para generar la derivada en cada punto se necesita información de varios puntos.

2. En segundo lugar se calcula sobre la matriz original, a partir de la matriz en la derivada, la integral correspondiente a la inversión de Abel.

$$f(r, y) = \frac{-1}{\pi} \int_r^{\infty} \frac{d\phi(x, y)}{dx} \frac{1}{\sqrt{x^2 - r^2}} dx$$

Esto se hace para cada punto de la imagen de salida, a una altura “y” y a una distancia “r” del centro de la imagen.

- Si el punto “r” es de la parte derecha de la imagen, se calcula la cantidad dentro de la integral para todos los valores de “x” mayores o iguales que el valor “r” y se suman.
- Si el punto “r” es de la parte izquierda de la imagen, se calcula la cantidad dentro de la integral para todos los valores de “x” menores o iguales que menos el valor “r” y se suman.
- Además, siempre hay una singularidad en  $x=r$ , de forma que para evitarla, se duplica el valor del integrando con el siguiente punto.

Esta operación por ser punto a punto se distribuye de igual forma que el ejemplo del apartado 4.6.3, pero también, como en el primer paso, el resultado en cada punto depende de una serie de puntos en la imagen original, por lo que es fundamental que la matriz de salida sea distinta que la de entrada.

#### 5.7.6. Extracción de una línea

La extracción de una línea no es más que un muestreo de puntos de la imagen a lo largo de un segmento. Para ello es necesario crear una nueva zona memoria para guardar los datos del segmento, que será el vector de salida.

Se realiza en dos pasos:

1. En primer lugar se determina el número de puntos de muestreo que va a contener la línea en función de su longitud. Para ello, si se va a extraer una

recta desde las coordenadas  $(x_1, y_1)$ , hasta  $(x_2, y_2)$ , el número de puntos que tendrá la muestra será:

$$n = \left\lfloor \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + 1 \right\rfloor$$

2. En segundo lugar se extrae información de los pixeles a lo largo de la línea, según el muestreo, de forma que el primer y último punto de la línea coincidan exactamente con los puntos inicial y final de la recta. Para ello, siendo  $j$  el índice que va desde 0 hasta  $n-1$ , y de cada bloque, se ejecutan  $n$  hilos en bloques de 32, idealmente en paralelo, cada uno de ellos encargado de llenar uno de los elementos del vector de salida, siendo la cuenta la siguiente:

$$x_{aux} = x_1 + \frac{(x_2 - x_1) \cdot j}{n - 1}$$

$$y_{aux} = y_1 + \frac{(y_2 - y_1) \cdot j}{n - 1}$$

$$salida[j] = phase[x_{aux}][y_{aux}]$$

#### 5.7.7. Resto de algoritmos

El resto de algoritmos, como ya se indicó en la introducción, consisten en simplemente aplicar operaciones punto a punto, y se distribuyen de igual forma que el ejemplo del apartado 4.6.3.

### **5.8. Secuencia de procesamiento**

Como ya se ha mencionado en los apartados anteriores, existen varios objetos por los que se va pasando para obtener el resultado final.

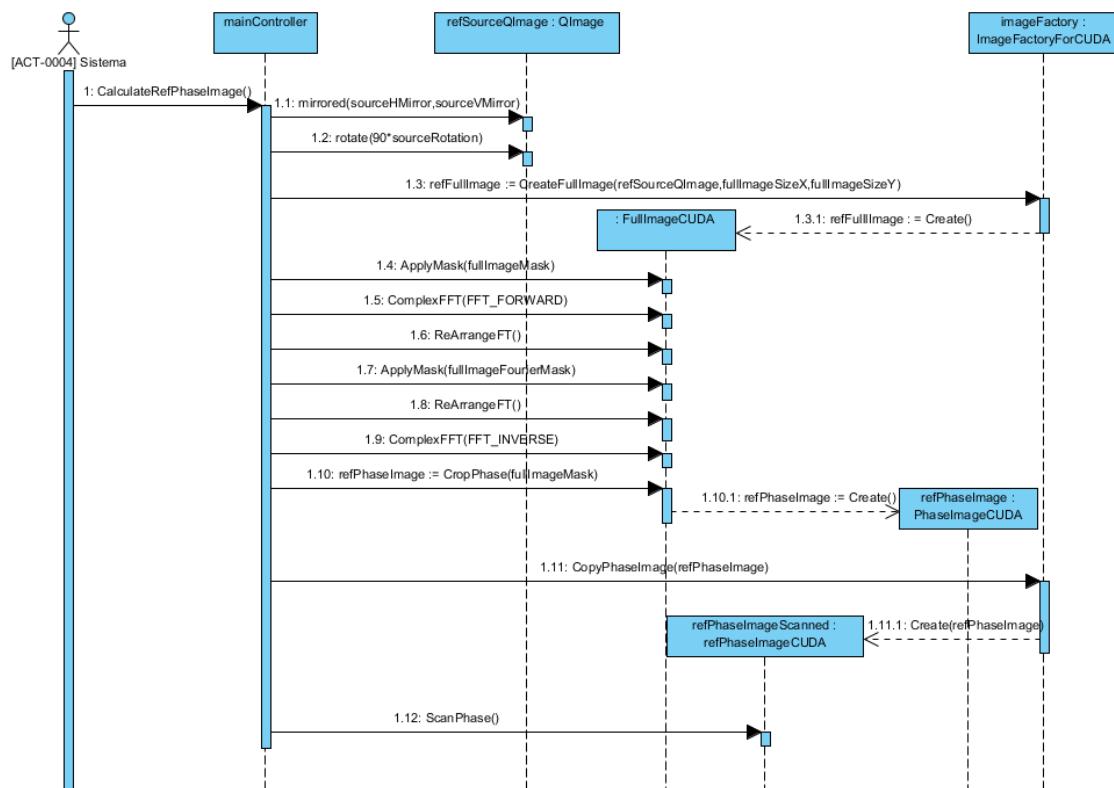
En este apartado se pretende mostrar cómo van cooperando los distintos objetos para lograr el objetivo final: obtener la imagen del jet procesada correctamente.

En los diagramas que se presentan se presupone que ya se ha creado una factoría de imágenes para CUDA, a la que el controlador principal tiene acceso, que se usará cuando proceda para la creación y copia de los objetos de imagen.

### 5.8.1. Obtención de la fase de referencia

En primer lugar, en la Figura 94, se observa la secuencia completa de obtención de la fase de referencia, que corresponde al proceso visto en el capítulo teórico 3.4 de acuerdo al procesado de la referencia.

Para ello, se supone que ya se ha cargado una imagen como referencia, y configurado unos parámetros.



*Figura 94. Diagrama de Secuencia: Procesado de la referencia*

Se puede observar cómo se crea a partir de la imagen un objeto del tipo FullImageCUDA, por medio de la factoría, sobre el que se realizan algunas operaciones y, más adelante, un objeto Phaselmage que, tras un breve procesado, contendrá la fase de la referencia.

Esta secuencia tendrá lugar cada vez que sea necesario obtener la fase de referencia desde cero por el motivo que sea.

### 5.8.2. Pre-procesado de una imagen

En segundo lugar, se pasa a pre-procesar una imagen, el proceso descrito en el capítulo teórico 3.4 de acuerdo al procesado de la imagen con la referencia.

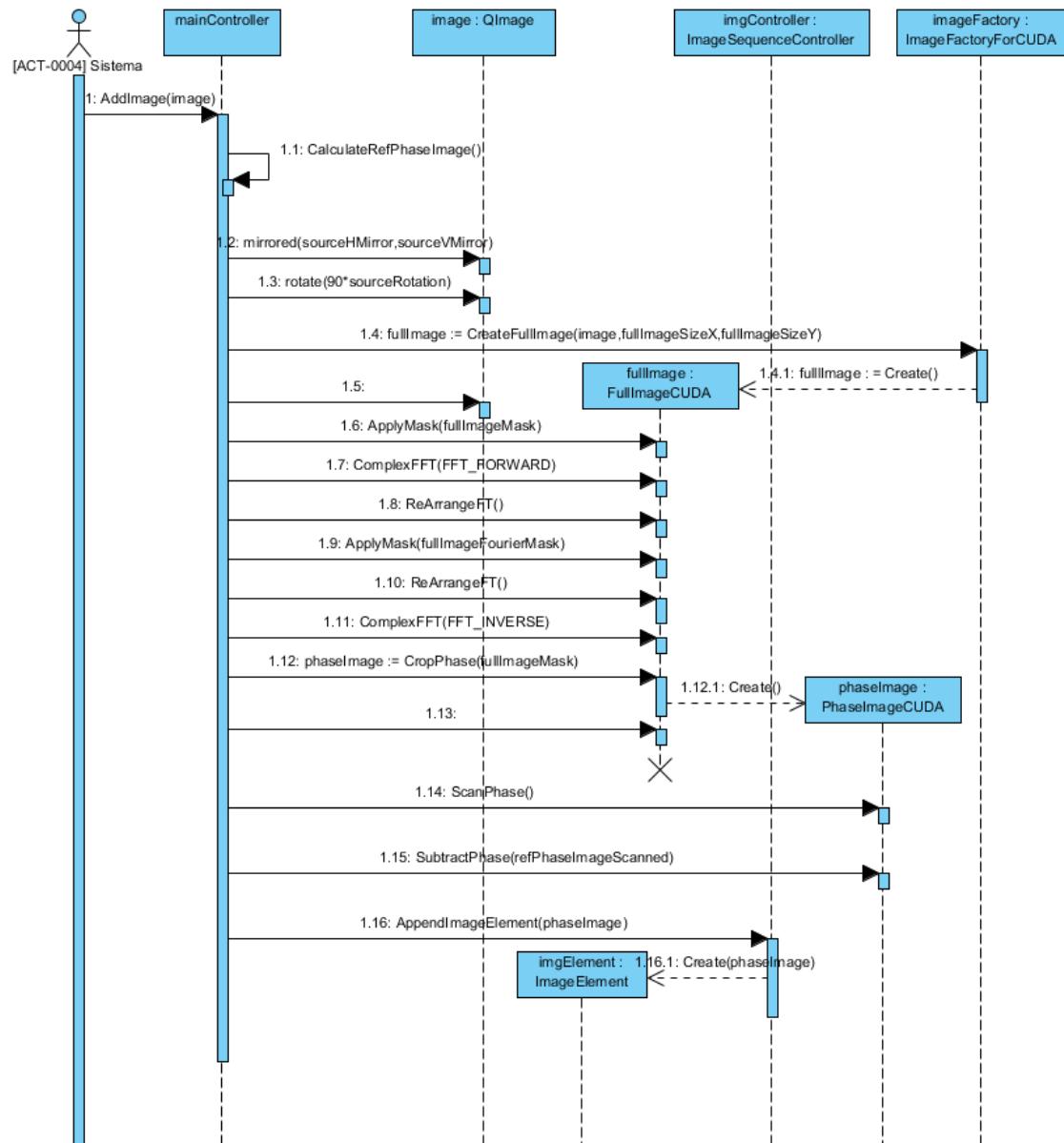


Figura 95. Diagrama de Secuencia: Pre-procesado de una imagen

Esto ocurre, por ejemplo, cuando al controlador le llega una petición de añadir una imagen de fase a partir de una imagen obtenida por algún medio.

El proceso, que se puede observar en la Figura 95, es prácticamente idéntico al de la obtención de la fase de referencia, con dos añadidos: ahora a la fase obtenida se le resta la fase de referencia, y además se añade a la lista de imágenes a través del controlador de la secuencia de imágenes.

### 5.8.3. Tratamiento posterior de la imagen de fase

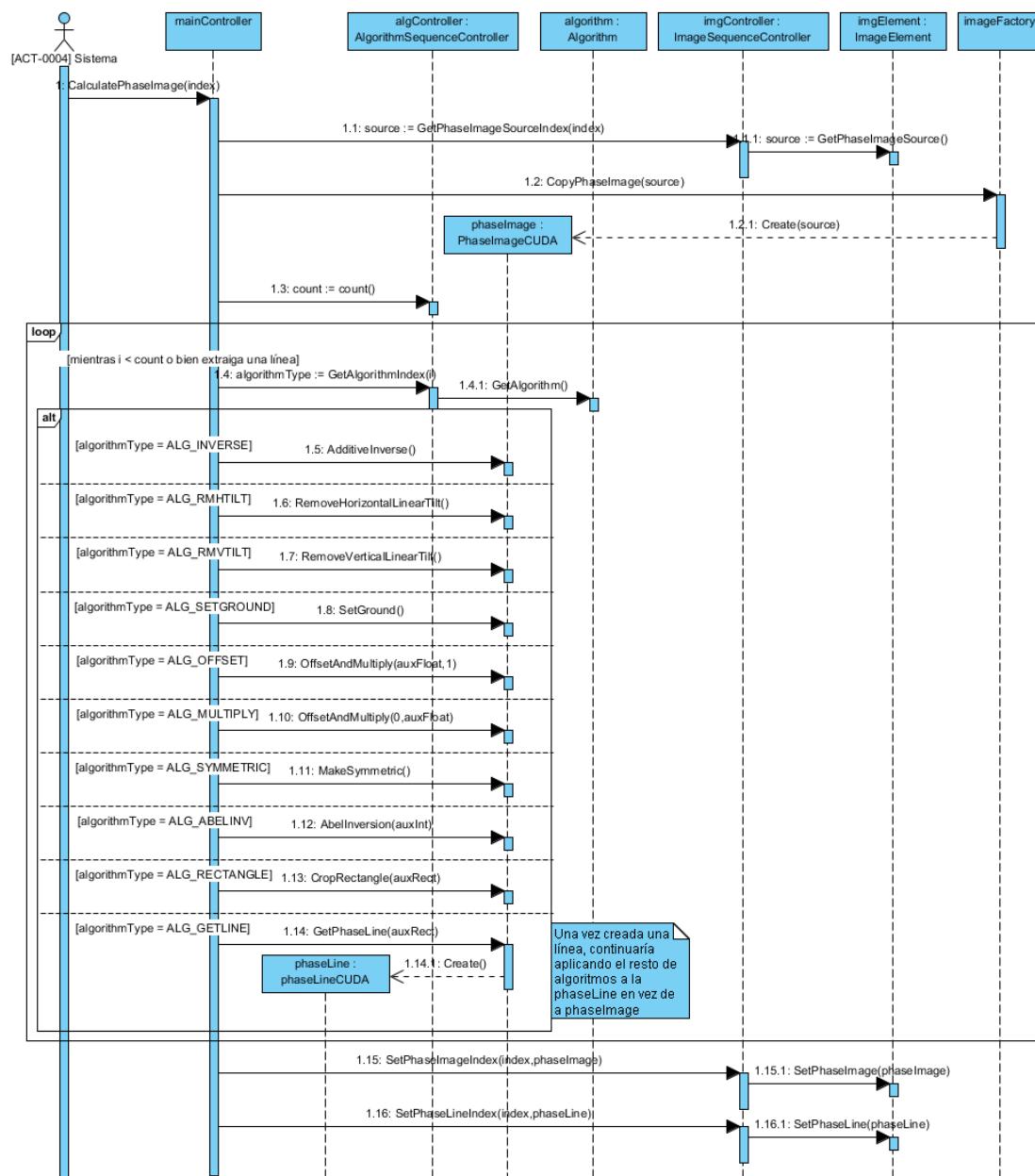


Figura 96. Diagrama de Secuencia: Aplicación de algoritmos

Consiste en la aplicación de la secuencia de algoritmos a la imagen de la fase, de acuerdo a lo explicado en el capítulo teórico 3.5.

En este proceso (Véase la Figura 96) el controlador principal obtiene una imagen de fase del controlador de imágenes, de la que crea una copia. A continuación va recuperando secuencialmente cada algoritmo a través de su controlador y aplicándolos a sobre la copia. Al final envía la copia procesada al controlador de imágenes para que la almacene en el lugar adecuado.

### **5.9. Política de cálculo**

Cuando se deben representar resultados que proceden de un cálculo previo hay en general dos opciones.

La primera forma de trabajar es la siguiente:

- Cada vez que se modifica algún parámetro se realizan de nuevo todos los cálculos.
- Para representar un resultado los datos simplemente se recuperan los resultados del cálculo.

Sin embargo esa forma de trabajar es muy costosa para nuestra aplicación, ya que implicaría, por ejemplo, aplicar la lista de algoritmos a todas las imágenes cada vez que se modifica un solo parámetro de esa lista, cuando muchas veces, realmente el resultado no nos interesaría hasta haber variado una serie de parámetros.

En resumen: al trabajar de esa forma se pueden llegar a producir muchos resultados que nunca van a ser utilizados.

Para evitar eso, la forma de calcular es la siguiente:

- Cada vez que se modifica algún parámetro que afecte al cálculo, se guarda.
- Para representar un resultado, se pide al controlador principal de la aplicación, el resultado de cierta imagen de la lista. El controlador sabrá entonces si ya

tiene actualizados los resultados para esa imagen, o si por el contrario los tiene que calcular, calculándolos si procede antes de devolver el resultado.

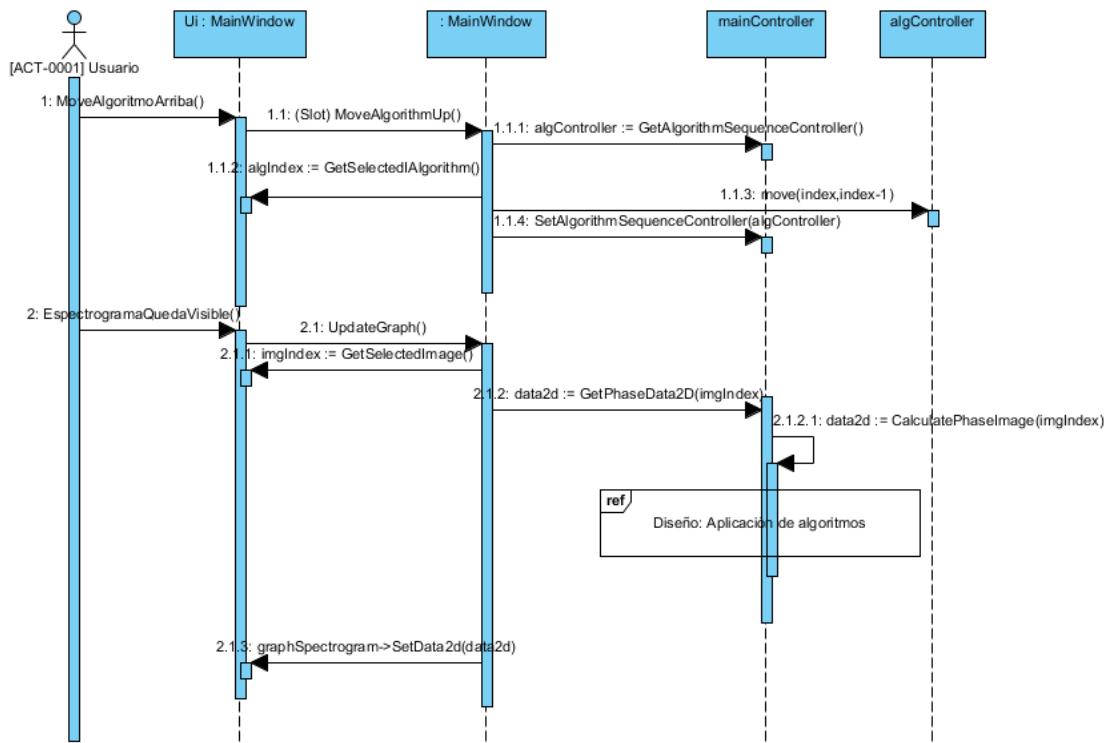


Figura 97. Diagrama de Secuencia: Gráficos en la ventana principal

Un ejemplo de esto en la interfaz principal cada vez que el usuario ha intercambiado la posición de dos algoritmos moviendo uno arriba, y acto seguido, se pasa a ver en la parte central el resultado de aplicar esa lista de algoritmos a la imagen de fase seleccionada. Como se puede observar en la Figura 97, la lista de algoritmos se actualiza, pero, sin embargo, sólo se aplica a la imagen que el usuario tiene seleccionada, con índice "imgIndex" en el gráfico, en vez de a toda la lista de imágenes.

### 5.10. Cámara y simulador de cámara: Factoría Abstracta

Como ya se ha comentado, una de las características de la aplicación consiste en tomar y procesar imágenes directamente de la cámara.

En este caso, las cámaras que se utilizaban, de IDS-Imaging, funcionaban con la API de µEye. No obstante, cada casa de sensores CCD tiene una API distinta, y es más que

probable, antes o después, un cambio de la cámara, y el consiguiente cambio necesario en el código.

Dado que eso es inevitable, se ha buscado una forma de que cuando eso ocurra, tenga las mínimas consecuencias para el programa, y es, de nuevo, la abstracción.



Figura 98. Clase de cámara abstracta

Para ello se especificó la clase **AbstractCamera**, que define una interfaz de operaciones habituales sobre una cámara. Estos métodos permiten tomar una simple imagen, variar parámetros como la ganancia, la exposición, el reloj de pixel, variar todos los parámetros de golpe a través de una clase creada para ellos (**CameraParameters**) o bien, comprobar si la cámara está actualmente abierta y disponible.

Para más información de este objeto abstracto consultese el Anexo IV de Manual del Programador.

En este caso hay dos clases que derivan de **AbstractCamera**:

- **CameraUEye:** Implementa AbstractCamera para las cámaras IDS-Imaging usadas en el laboratorio con la API μEye.
- **CameraSimulator:** Se utiliza cuando no se tiene una cámara disponible y se quiere observar la funcionalidad de la aplicación relacionada con la cámara. Para ponerlo a funcionar es necesario abrir el programa con el acceso directo que indica (*With Camera Simulator*), o bien, lanzarlo por línea de comandos con el argumento “-cs” o “-CameraSimulator”. Esta clase implementa los tiempos de espera por la exposición como si de una cámara real se tratase, y varía coherentemente los valores máximos y mínimos de unos parámetros en función de los otros. Las imágenes que devuelve como tomadas, las obtiene de un directorio “CameraSimulator” en la propia carpeta del programa ejecutable.

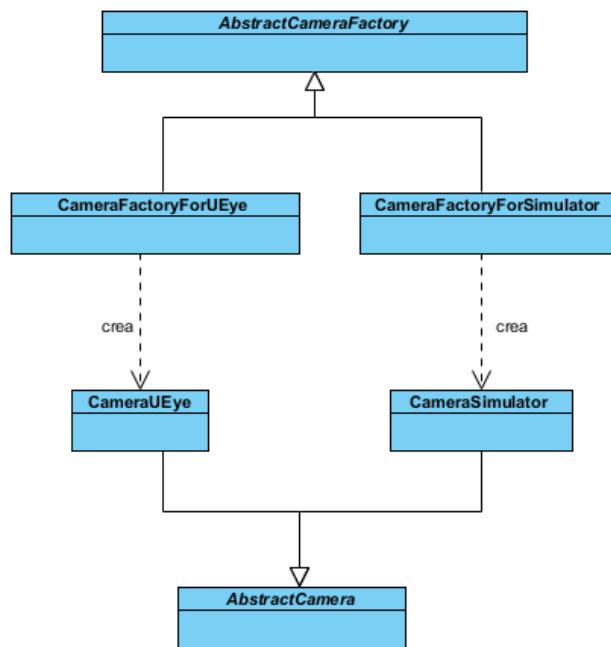


Figura 99. Abstracción y factorías de la cámara

Un aspecto a tener en cuenta es que la cámara es hardware externo al ordenador, y por tanto puede ser conectado o desconectado durante la ejecución del programa. Por ello, la aplicación está constantemente comprobando si se ha conectado una cámara, o si la cámara que había conectada se ha desconectado y eso puede llegar a implicar la creación y destrucción de muchos objetos de cámara.

Dado que siempre se busca que la cámara para la que se está consultando la conexión sea del mismo tipo, se recurre de nuevo al uso de factorías, pudiendo, una vez iniciada la factoría, crear y destruir las cámaras con independencia del tipo de cámara.

El diagrama de clases para la factoría es el que se puede observar en la Figura 99, y la idea de funcionamiento es la misma que la que se encuentra en el apartado 5.4.

### 5.11. Toma de imágenes de forma eficaz

Uno de los requisitos no funcionales del que ya se ha hablado bastante, es la capacidad de la aplicación de mostrar el jet a tiempo real lo más rápido posible. Hasta ahora se ha hablado de la parte del procesado la imagen, usando procesamiento en paralelo para su optimización. Sin embargo hay otro punto crítico en el proceso: la toma de la imagen.

Típicamente para el tipo de experimento que se está realizando, la imagen se toma con una exposición de unos 100 ms, y durante esos 100 ms el sistema se queda esperando a que la célula CCD acabe de recoger fotones, es decir, bloqueado sin consumo de CPU desde que pide la imagen hasta que la obtiene.

Como en este caso no se puede permitir tan grandísima pérdida de tiempo, se ha recurrido a la utilización de un hilo en paralelo que se encargue de tomar las imágenes.

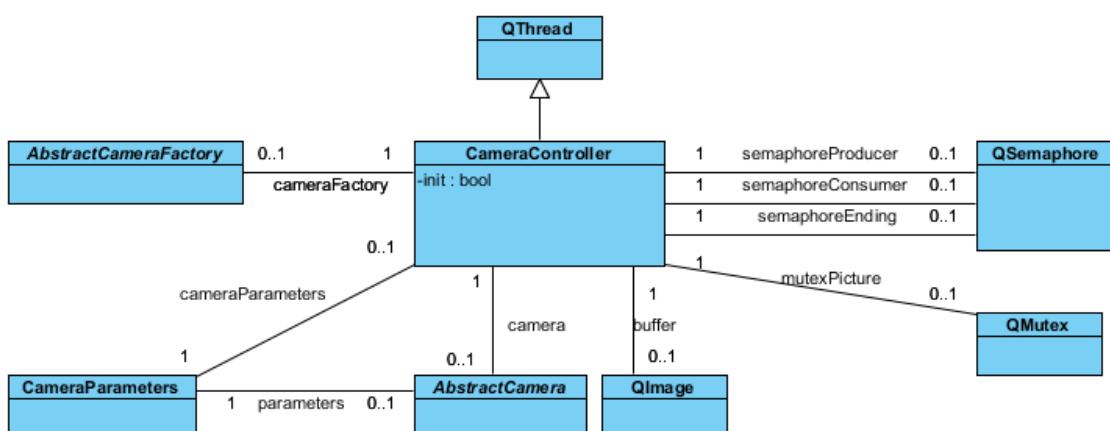


Figura 100. Controlador de cámara y sus relaciones a segundo nivel

El objetivo es que el controlador de la cámara (CameraController), tenga siempre disponible una imagen cuando se la pidan de forma que mientras se está procesando una imagen, un hilo que ejecute el controlador ya esté bloqueado tomando la siguiente.

El método del controlador que se ejecutará en el otro hilo será el método “run”, que además comenzará a funcionar a partir de que se llame al método “initialize” del mismo controlador.

Sin embargo, una limitación que tendría esta técnica es que no garantiza que la imagen sea reciente cuando lo que se está tomando es una sola imagen, y no una secuencia (En el caso de una secuencia puede que la primera no lo fuera, pero el resto sí). La solución a esto es incluir un método que permita renovar la imagen “RenewPicture” de forma que cuando sea llamado, el controlador deseche la imagen anterior y tome una nueva.

La forma de coordinar estos dos hilos se realiza mediante el uso de semáforos. En este caso se utilizan 3 semáforos y un mutex. Dos de los semáforos se utilizan para controlar el flujo productor-consumidor, y el tercer semáforo para asegurar una finalización ordenada del hilo, al destruir el controlador. El mutex simplemente se utiliza para evitar que se modifiquen parámetros de la cámara mientras se está tomando una imagen.

El esquema de utilización de los semáforos es el siguiente (Sólo se muestra la parte del código relacionada con los semáforos):

Por una parte al iniciar el hilo (Con “start”):

```
bool CameraController::Initialize() {
    semaphoreProducer = new QSemaphore(0);
    semaphoreConsumer = new QSemaphore(0);
    semaphoreEnding = new QSemaphore(0);
    mutexPicture = new QMutex();
    //Libero el semáforo del productor
    semaphoreProducer->signal();
```

```
    this->start();
}
```

El hilo encargado de tomar la imagen:

```
for(;;) {
    //Espero a que el cliente consuma
    semaphoreProducer->wait(INFINITE);
    //Si me han dicho que tengo que finalizar salgo del bucle
    if(semaphoreEnding->wait (0)==true) break;
    //Tomo la imagen
    mutexPicture->wait(INFINITE);
    buffer=camera->AcquireSourcePicture();
    mutexPicture->signal();
    //Aviso al cliente que ya hay disponible
    semaphoreConsumer->signal();
}

```

El método que renueva la imagen (Sólo tiene sentido renovarla si había una disponible):

```
if(semaphoreConsumer->wait(0)==true) {
    delete buffer;
    buffer=NULL;
    semaphoreProducer->signal();
}
```

El método que permite obtener la imagen:

```
semaphoreConsumer->wait(INFINITE);
aux=buffer;
buffer=NULL;
semaphoreProducer->signal();
```

Un método cualquiera que varía un parámetro de la cámara:

```
mutexPicture->wait(INFINITE);
returnValue=camera->SetGain(command,value);
mutexPicture->signal();
```

Un método que interrumpe el hilo:

```
//Libero el semáforo de finalizar para que el hilo acabe
semaphoreEnding->signal();

//Libero el semáforo del productor, para asegurar que no se
//quedá atascado en ese paso
semaphoreProducer->signal();

//Espero a que el hilo haya acabado (Al acabar envía
//una señal al controlador)
this->wait();
```

Y por último, obsérvese en la Figura 101 cómo funcionaría la parte productor consumidor (Sin considerar el mutex o la finalización por simplicidad) en un diagrama de secuencia.

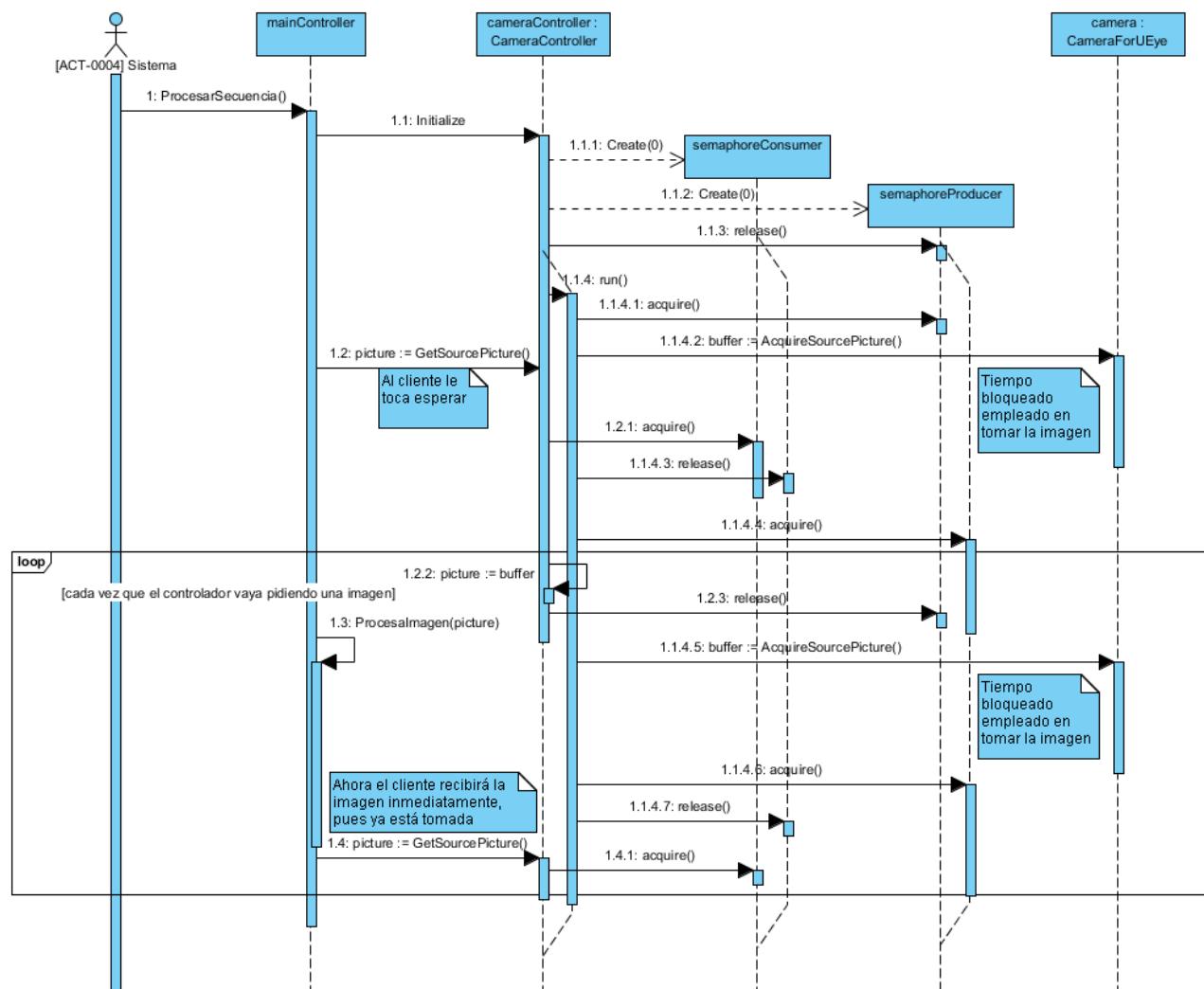


Figura 101. Diagrama de Secuencia: Hilo de toma de imágenes

### **5.12. Modo “en vivo”: Mejora de la experiencia del usuario**

El modo en vivo consiste en la toma continua de imágenes y su procesado con la fase de referencia.

La forma más sencilla de realizar eso, sería que el controlador de la interfaz pidiera al controlador principal que procesara una imagen, el controlador principal la devolviera, y la interfaz la mostrara.

No obstante en esta aplicación, para la que se busca una frecuencia de procesamiento lo más alta posible, esa forma de trabajar no es viable, ya que la mayor parte del tiempo tendría el control el controlador principal realizando el procesamiento, y el controlador de la interfaz se quedaría bloqueado, recibiendo el control tan solo durante unos pocos milisegundos cada segundo, lo que haría imposible recibir y procesar acciones del usuario de forma agradable para la experiencia de éste, como por ejemplo girar suavemente un gráfico 3D.

La forma de evitar ese problema es, de nuevo, recurrir a los hilos.

La secuencia consiste en:

- El controlador de la interfaz establece un temporizador para representar una imagen.
- Cuando llega una llamada del temporizador, un semáforo indicará si hay algo procesándose:
  - Si no se está procesando nada, se crea una instancia en otro hilo de la clase DataProducer, que mandará al controlador principal que tome una imagen y la procese.
  - Si se está procesando algo se programa otro temporizador para dentro de unos segundos: sólo se quiere estar procesando una cosa a la vez.
- Cuando el hilo que procesa acaba, envía una señal al controlador de la interfaz avisando de que ya tiene los datos, y se destruye. El controlador de la interfaz recibe la señal y representa los datos. Después comienza el ciclo de nuevo.

Este comportamiento se puede ver en la Figura 102.

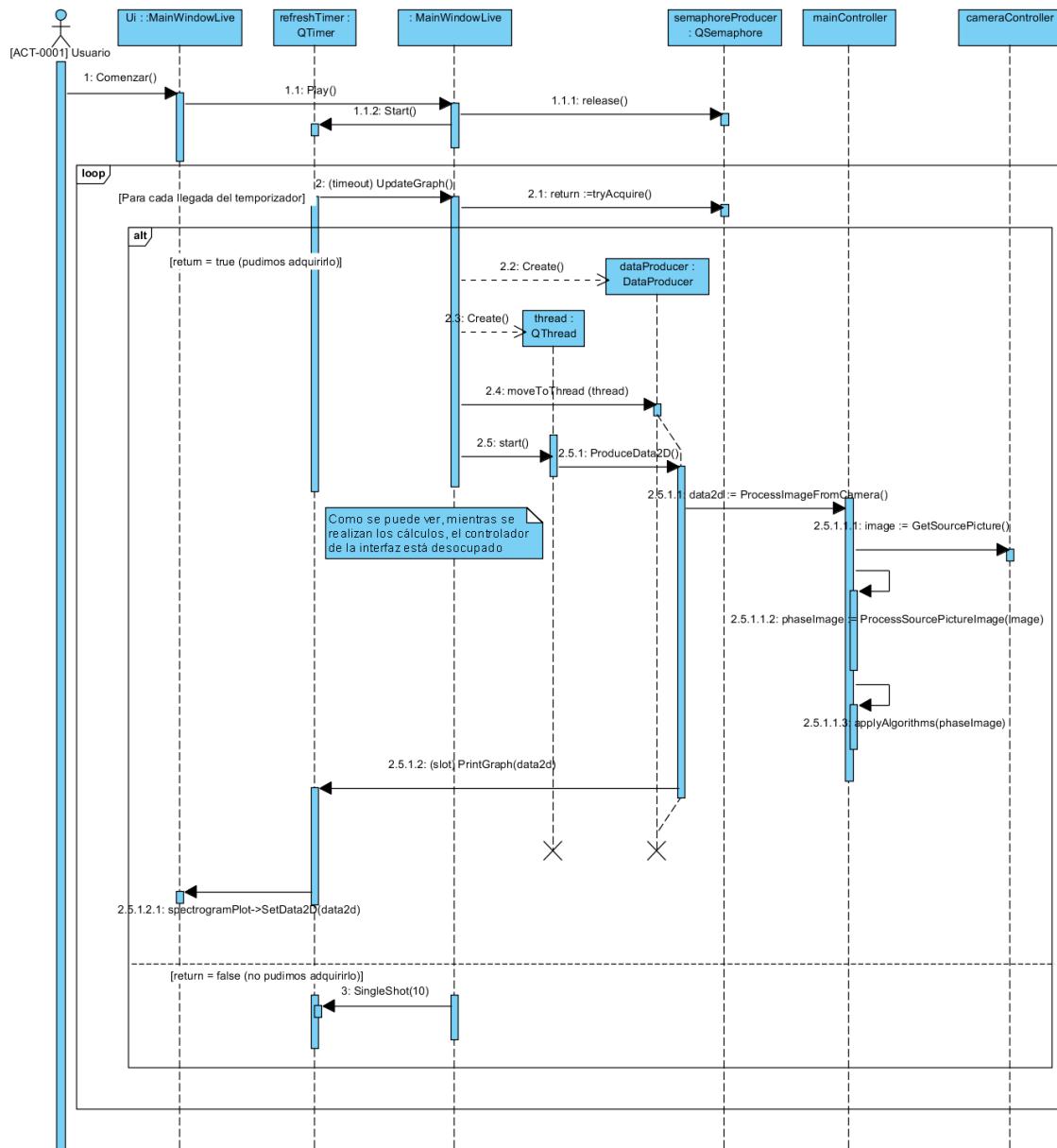


Figura 102. Diagrama de Secuencia: Procesado de imagen en modo “en vivo”

Como se puede comprobar, a partir de que el controlador de la interfaz llama a `start()`, el procesamiento ocurre en un hilo diferente, y el controlador de la interfaz está libre para recibir acciones del usuario, estando ocupando sólo durante el tiempo necesario para la representación del gráfico, tiempo lo suficientemente pequeño como para que no afecte a la experiencia del usuario.

### **5.13. Guardado en ficheros**

Una de las características más cómodas de la aplicación, es que casi todos los conjuntos de parámetros se pueden guardar en ficheros, de forma que se facilita su reutilización, evitando la introducción manual de los mismos parámetros, por ejemplo de la cámara, cada vez que se va a realizar un experimento parecido al anterior que requiera la misma configuración.

Como veremos algunos de los siguientes archivos se utilizan para que el usuario pueda cargarlos y guardarlos, otros como almacenamiento persistente de preferencias de la aplicación en la carpeta de configuración local de programa ubicada en la zona indicada por el sistema operativo para ello, y otros se utilizan para conformar la entidad llamada “proyecto”.

#### **5.13.1. Máscaras, parámetros y opciones de visualización**

Para el guardado de máscaras, operaciones y opciones de visualización, dado que no se necesita guardar listas, sino un conjunto conocido de valores, se optó por los archivos INI.

Existe el archivo con extensión “.msk1” que representa la máscara directa junto con el origen de coordenadas. Es algo de la forma:

```
[General]
OriginX=588
OriginY=304
x1=492
x2=684
y1=304
y2=580
```

También el archivo con extensión “.msk2” que representa la máscara de Fourier, como por ejemplo:

```
[General]
x1=1001
```

```
x2=1055  
y1=428  
y2=465
```

El archivo de operaciones, con extensión “.ops”, que guarda otros parámetros y operaciones que se aplican a la referencia:

```
[General]  
SourceHMirror=false  
SourceVMirror=false  
SourceRotation=0  
FullImageSizeX=2048  
FullImageSizeY=1024  
PixelsMmX=@Variant(\0\0\0\x87@\xa0\0\0)  
PixelsMmY=@Variant(\0\0\0\x87@\xa0\0\0)
```

Nótese que ese formato extraño usado para PixelsMmX y PixelsMmY es la forma que utiliza Qt con su clase QSettings, usada para generar estos archivos, de guardar los números en coma flotante con el formato Variant.

Y por último el archivo de opciones de visualización, con extensión “.viw”, que guarda parámetros relacionados con las opciones de visualización disponibles en la aplicación:

```
[General]  
ZLabel=Phase  
AutoScaleAxes=false  
MinZValue=@Variant(\0\0\0\x87\0\0\0\0)  
MaxZValue=@Variant(\0\0\0\x87?\xa6\x66\x66)  
SequenceFPS=@Variant(\0\0\0\x87\x41\xf0\0\0)  
ViewLevelCurves=false  
Threshold=@Variant(\0\0\0\x87?333)  
ApplyAlgorithmsLive=true  
AutoScaleAxesLive=false  
ThresholdLive=@Variant(\0\0\0\x87?333)  
LiveFPS=@Variant(\0\0\0\x87@\xa0\0\0)
```

### 5.13.2. Lista de algoritmos

Para la lista de algoritmos, por su naturaleza de lista, se ha utilizado XML, creando un archivo con extensión “.alg”.

Se trata de almacenar una lista de objetos de tipo Algorithm, para lo que se crea la siguiente estructura de archivo:

-Utiliza un tipo de documento llamado AlgorithmList

```
<!DOCTYPE AlgorithmList>
```

-Y crea un primer nivel llamado también AlgorithmList.

```
<AlgorithmList>
```

-Dentro de este primer nivel se encuentra un primer elemento, un número, la cuenta del número de algoritmos:

```
<Count>4</Count>
```

-Y a continuación una serie de nodos tipo Algorithm conteniendo cada uno de ellos el tipo y todos los parámetros del algoritmo asociado:

```
<Algorithm>
  <AlgorithmType>3</AlgorithmType>
  <AuxInt>0</AuxInt>
  <AuxFloat>0</AuxFloat>
  <AuxRectX1>42</AuxRectX1>
  <AuxRectX2>155</AuxRectX2>
  <AuxRectY1>20</AuxRectY1>
  <AuxRectY2>257</AuxRectY2>
</Algorithm>
```

Para terminar veamos un ejemplo de resultado con 4 algoritmos:

```
<!DOCTYPE AlgorithmList>
<AlgorithmList>
  <Count>4</Count>
  <Algorithm>
```

```
<AlgorithmType>0</AlgorithmType>
<AuxInt>0</AuxInt>
<AuxFloat>0</AuxFloat>
<AuxRectX1>0</AuxRectX1>
<AuxRectX2>-1</AuxRectX2>
<AuxRectY1>0</AuxRectY1>
<AuxRectY2>-1</AuxRectY2>
</Algorithm>
<Algorithm>
<AlgorithmType>8</AlgorithmType>
<AuxInt>0</AuxInt>
<AuxFloat>0</AuxFloat>
<AuxRectX1>42</AuxRectX1>
<AuxRectX2>155</AuxRectX2>
<AuxRectY1>20</AuxRectY1>
<AuxRectY2>257</AuxRectY2>
</Algorithm>
<Algorithm>
<AlgorithmType>1</AlgorithmType>
<AuxInt>0</AuxInt>
<AuxFloat>0</AuxFloat>
<AuxRectX1>42</AuxRectX1>
<AuxRectX2>155</AuxRectX2>
<AuxRectY1>20</AuxRectY1>
<AuxRectY2>257</AuxRectY2>
</Algorithm>
<Algorithm>
<AlgorithmType>9</AlgorithmType>
<AuxInt>0</AuxInt>
<AuxFloat>0</AuxFloat>
<AuxRectX1>6</AuxRectX1>
<AuxRectX2>110</AuxRectX2>
<AuxRectY1>60</AuxRectY1>
<AuxRectY2>60</AuxRectY2>
</Algorithm>
</AlgorithmList>
```

### 5.13.3. Lista de imágenes

Para la lista de imágenes, también por ser una lista de objetos `PhaselImage`, se ha optado por el uso de XML. Sin embargo en este caso, es un poco más complicado, ya que además de parámetros sueltos para cada imagen es necesario guardar el trozo de memoria con los datos de la imagen, que es candidato a ser guardado en un archivo binario.

Para compaginar esas dos situaciones, se crean dos archivos, uno con toda la información de los atributos de los objetos `PhaselImage` en formato XML, y con extensión “.iml”, y otro en formato binario “.dat”, con los búferes de todas las imágenes, de forma que en el fichero XML exista una referencia al nombre del archivo binario.

De esta forma la estructura del archivo XML:

-Utiliza un tipo de documento llamado `ImageList`

```
<!DOCTYPE ImageList>
```

-Y crea un primer nivel llamado también `ImageList`.

```
<ImageList>
```

-Dentro de este primer nivel se encuentra la cuenta del número de imágenes:

```
<Count>3</Count>
```

-Ahora se incluye la referencia al archivo de datos:

```
<DataFile>ImageData.dat</DataFile>
```

-Y a continuación una serie de nodos tipo `Image` contenido cada uno de ellos todos los parámetros de la imagen asociada, incluyendo los parámetros `SizeX` y `SizeY`, de forma que cada vez que se lea una nueva imagen, se sabe que se tiene que leer un buffer de dimensiones tamaño “`SizeX·SizeY`” del archivo de datos binarios:

```
<Image>
```

```
<Label>Uno</Label>
<PixelsFromOriginX>-96</PixelsFromOriginX>
<PixelsFromOriginY>0</PixelsFromOriginY>
<PixelsMmX>5</PixelsMmX>
<PixelsMmY>5</PixelsMmY>
<SizeX>193</SizeX>
<SizeY>277</SizeY>
</Image>
```

Y un archivo completo de ejemplo con tres imágenes queda de la siguiente forma:

```
<!DOCTYPE ImageList>
<ImageList>
  <Count>3</Count>
  <DataFile>ImageData.dat</DataFile>
  <Image>
    <Label>Uno</Label>
    <PixelsFromOriginX>-96</PixelsFromOriginX>
    <PixelsFromOriginY>0</PixelsFromOriginY>
    <PixelsMmX>5</PixelsMmX>
    <PixelsMmY>5</PixelsMmY>
    <SizeX>193</SizeX>
    <SizeY>277</SizeY>
  </Image>
  <Image>
    <Label>Dos</Label>
    <PixelsFromOriginX>-96</PixelsFromOriginX>
    <PixelsFromOriginY>0</PixelsFromOriginY>
    <PixelsMmX>5</PixelsMmX>
    <PixelsMmY>5</PixelsMmY>
    <SizeX>193</SizeX>
    <SizeY>277</SizeY>
  </Image>
  <Image>
    <Label>Tres</Label>
    <PixelsFromOriginX>-96</PixelsFromOriginX>
    <PixelsFromOriginY>0</PixelsFromOriginY>
    <PixelsMmX>5</PixelsMmX>
    <PixelsMmY>5</PixelsMmY>
    <SizeX>193</SizeX>
  </Image>
</ImageList>
```

```
<SizeY>277</SizeY>
</Image>
</ImageList>
```

#### 5.13.4. Preferencias del programa y parámetros de la cámara

Las preferencias del programa y parámetros de la cámara, se guardan también en sendos archivos INI, en este caso manteniendo la extensión “.ini”.

El archivo de preferencias del programa es algo de la forma:

```
[General]
Mode=1
UseDefaultMode=false
OutputFolder=D:/SalidaJetProcessing
SaveCameraPictures=false
ApplyUntilCurrentAlgorithm=true
SaveGraphicsViewSize=false
CustomSize=@Size(600 600)
DefaultImageFormat=png
FirstExecution=false
```

Mientras que el archivo de parámetros de la cámara tiene la apariencia:

```
[General]
SizeX=1280
SizeY=1024
PixelClock=24
FrameRate=9.00268279947424
Exposure=12.79308333333333
MasterGain=0
RedGain=10
GreenGain=0
BlueGain=15
```

Cada vez que se cierre el programa, se guardarán dos archivos con los nombres “settings.ini” y “camerasettings.ini” en una carpeta creada en el directorio del sistema operativo indicada para guardar archivos de configuración local.

De esta forma, en la siguiente ejecución, es posible continuar con la misma configuración.

### 5.13.5. Guardado de proyecto

La clave acerca del guardado del proyecto está en torno a la cuestión: ¿qué conforma un proyecto?

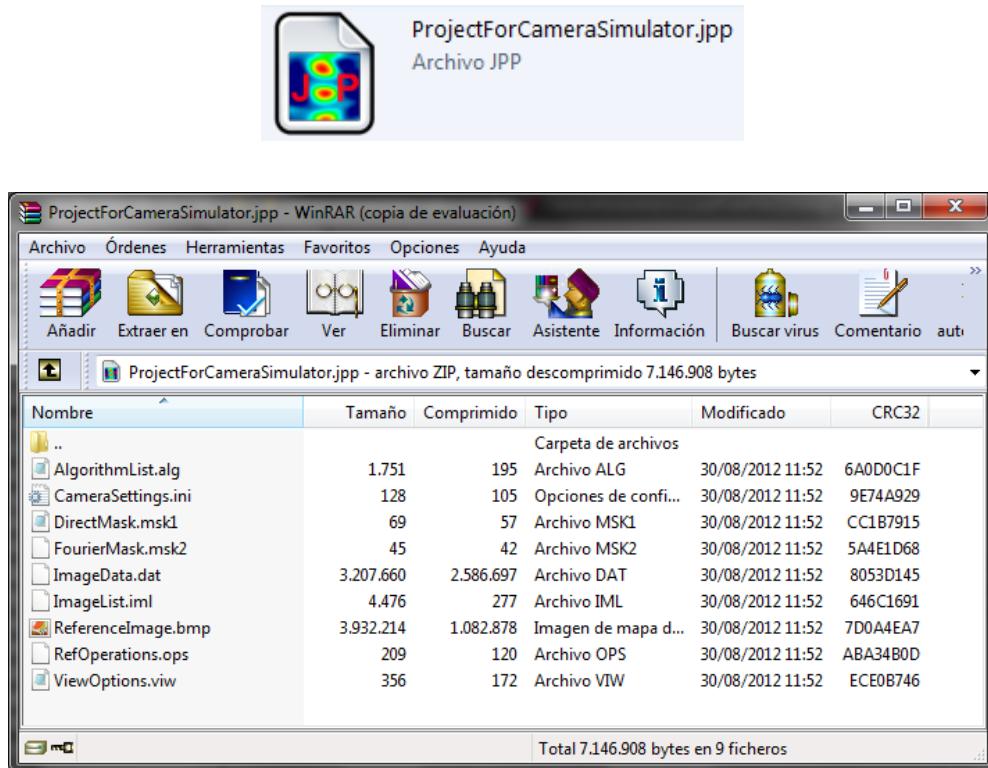
En la aplicación JetProcessing se considera parte del proyecto a los elementos centrales que se han cargado o fijado a lo largo de la configuración del programa, salvo las preferencias del programa, que se consideran parámetros del programa y no del proyecto.

Esto hace que guardar un proyecto, sea sinónimo de guardar la imagen de la referencia, las operaciones realizadas sobre la referencia, las máscaras utilizadas para la referencia, la lista de algoritmos, la lista de imágenes, la configuración de la cámara y las opciones de visualización activadas.

Por lo tanto, vistos los apartados anteriores un proyecto constará de 9 archivos:

-ReferencelImage.bmp	Imagen de la referencia guardada sin pérdida.
-RefOperations.ops	Operaciones y parámetros para la referencia.
-DirectMask.msk1	Máscara directa y origen de coordenadas.
-FourierMask.msk2	Máscara de Fourier.
-AlgorithmList.alg	Lista de algoritmos.
-ImageList.iml	Lista de imágenes.
-ImageData.dat	Búferes de con los datos de la lista de imágenes.
-CameraSettings.ini	Parámetros de la cámara.
- ViewOptions.viw	Opciones de visualización seleccionadas.

Sin embargo, no es cómodo trabajar con 9 archivos sueltos, por lo que todos ellos se crearán en una carpeta temporal que después se comprimirá en un fichero comprimido ZIP marcado con la extensión “.jpp” (JetProcessing Project).



*Figura 103. Archivo de proyecto*

A continuación para usar ese fichero, el programa realizará la operación contraria: lo descomprimirá en una carpeta temporal, y leerá los distintos archivos.

Esta característica se puede ver reflejada en la Figura 103.

#### 5.13.6. Exportado de datos

Además del exportado de los gráficos con las imágenes calculadas, casi siempre es fundamental tener acceso a los propios datos del gráfico, para que puedan ser utilizados con otros programas como MATLAB® o Wolfram Mathematica®, o incluso comprobados manualmente, pudiendo ver el valor asociado a un punto de la imagen.

La forma de permitir eso en JetProcessing es el uso de archivos de texto plano para exportar las imágenes como datos.

Dado que existen dos tipos de resultados, línea de fase, e imagen de fase, habrá dos formatos para los archivos de salida.

Para la línea de fase, objeto tipo Data1D, la primera columna representa la variable independiente y la segunda la variable dependiente, quedando el archivo:

-2.600354e+00	6.966785e-01
-2.400327e+00	7.082317e-01
-2.200300e+00	7.188846e-01
-2.000272e+00	7.286259e-01
-1.800245e+00	7.374325e-01
-1.600218e+00	7.452854e-01
-1.400191e+00	7.521675e-01
-1.200163e+00	7.580559e-01
-1.000136e+00	7.629296e-01
.	
.	
.	

Para la imagen de fase, objeto tipo Data2D, la primera columna representa la variable independiente “X”, la segunda la variable independiente “Y” y la tercera la variable dependiente, quedando el archivo:

-9.791151e+00	-5.160000e+01	4.154086e-03
-9.589380e+00	-5.160000e+01	1.488143e-02
-9.387610e+00	-5.160000e+01	2.564687e-02
-9.185841e+00	-5.160000e+01	3.645432e-02
-8.984071e+00	-5.160000e+01	4.729223e-02
-8.782301e+00	-5.160000e+01	5.817211e-02
-8.580531e+00	-5.160000e+01	6.907496e-02
.		
.		
.		
-9.791151e+00	-5.139915e+01	3.747642e-03
-9.589380e+00	-5.139915e+01	1.432240e-02
-9.387610e+00	-5.139915e+01	2.493906e-02
-9.185841e+00	-5.139915e+01	3.559011e-02
-8.984071e+00	-5.139915e+01	4.627544e-02
-8.782301e+00	-5.139915e+01	5.699128e-02
-8.580531e+00	-5.139915e+01	6.774148e-02
.		
.		

De esa forma, por ejemplo, un simple script de MATLAB, que muestra un espectrograma 3D con los datos procesados en JetProcessing y exportados en un archivo, es el siguiente:

```
% Carga los datos y extrae la información x y z
mat=load ('data2d.txt','-ascii');

% Determina los valores máximos y mínimos:
xmin = min(mat(:,1)); ymin = min(mat(:,2));
xmax = max(mat(:,1)); ymax = max(mat(:,2));
% Define la resolución para representar
xres=100;yres=100;
% Define el rango de espaciado de las coordenadas x-y,
xv = linspace(xmin, xmax, xres);
yv = linspace(ymin, ymax, yres);
[Xinterp,Yinterp] = meshgrid(xv,yv);

% Calcula Z en los puntos X-Y interpolando los valores
% adecuadamente
Zinterp = griddata(mat(:,1),mat(:,2),mat(:,3),Xinterp,Yinterp);
% Genera el espectrograma
surf(Xinterp,Yinterp,Zinterp);
```

Obteniendo un resultado como el que se puede observar en la Figura 104.

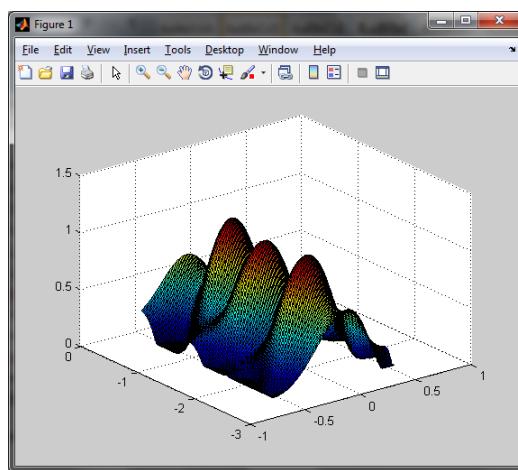


Figura 104. Gráfico generado cargando los datos en MATLAB

### 5.13.7. Exportado de vídeos y animaciones

En JetProcessing, también es posible exportar una secuencia de imágenes como un vídeo en formato AVI o una animación en formato GIF. Dado que no fue fácil encontrar bibliotecas multiplataforma para Qt que realizaran esto, se optó, por usar herramientas externas gratuitas multiplataforma que se incluyeran en el instalador del programa, y que fueran capaces de, a través de una orden por línea de comandos, transformar un conjunto de imágenes en un vídeo /animación. Estas herramientas son las indicadas en los apartados 4.9 y 4.10.

Para crear el vídeo/animación, se siguen los siguientes pasos:

1. Se exportan todos los archivos con los gráficos a una carpeta en un directorio temporal del sistema operativo.
2. Se lanza un proceso a través de QProcess con el comando correspondiente para que la aplicación externa cree el archivo de salida con el vídeo/animación en la carpeta indicada.
3. Se comprueba que el proceso finalizó correctamente.
4. Se vacía la carpeta temporal.

De esta forma, a pesar de trabajar con una herramienta externa, la dificultad que ello conlleva queda limitada a la aplicación, y el usuario no percibe ninguna diferencia respecto de cuando exporta, por ejemplo, una imagen como gráfico, o como datos en texto plano.

Nótese que a partir de 50 imágenes la opción de exportar como animación se desactiva, por no ser ya un formato adecuado.

## ***5.14. Detalles para un uso más cómodo***

Con JetProcessing se pretendía realizar una aplicación que no sólo cumpliera unos requisitos funcionales y con un buen rendimiento, sino que además ofreciera una buena experiencia al usuario, no tanto en los efectos visuales, dado que en una

aplicación científica como esta tienen escaso valor, sino en la comodidad de uso, evitando al usuario en la medida de lo posible realizar operaciones repetitivas, introducir varias veces los mismos parámetros, o tener que dedicar demasiado tiempo cada día a colocar el programa en la situación en la que lo dejó el día anterior.

Para lograr ese objetivo se han implementado una serie de características que, aunque no aportan funcionalidad de cálculo, son muchas veces el factor detonante que hace que un programa sea útil en la vida real.

Algunas de esas características son las siguientes:

- **Proyecto:** Todos los parámetros e información más importante que utiliza el programa como conjunto, se puede guardar en un archivo de proyecto, de forma que el trabajo se pueda interrumpir en cualquier momento y ser retomado al día siguiente.
- **Asociación del sistema operativo:** En el momento de la instalación, los archivos con extensión “.jpp” se asocian al ejecutable del programa, de forma que un proyecto se puede abrir simplemente haciendo doble-click sobre él. Además se asigna a los archivos de proyecto un ícono especialmente creado para ellos.
- **Gráficos interactivos:** Los gráficos que se visualizan, son interactivos para el usuario, permitiendo realizar operaciones sobre ellos a tiempo real, de forma suave para el usuario, sin trompicones, y que además se mantengan durante toda la secuencia de imágenes.
- **Exportado:** Los resultados se pueden guardar como datos numéricos, imagen, vídeo, animación..., permitiendo al usuario, además, elegir el tamaño de salida y el formato por defecto.
- **Guardado de parámetros:** Casi todos los conjuntos de parámetros se pueden guardar en archivos con extensión propia, para su uso en futuros proyectos, y así evitar en la medida de lo posible realizar el mismo trabajo dos veces.
- **Carpeta de trabajo:** El programa permite definir una carpeta de trabajo, de forma explícita en la primera ejecución, o a través del menú en las siguientes, donde se crearán los archivos de exportado de series completas, creando un

directorio exclusivo para cada serie, con un nombre elegido por el usuario. Así esta carpeta ayudará a centralizar todos los archivos generados con JetProcessing.

- **Múltiple acceso a la funcionalidad:** Existe un menú desde el que se puede acceder a todas las opciones del programa. Además existe una barra de herramientas y diferentes botones para facilitar el acceso a la funcionalidad más habitual. Por supuesto, también es posible controlar el programa mediante teclado.
- **Fijación de máscaras:** A pesar de ser una aplicación científica, se han incluido características de usabilidad como la posibilidad de tomar las máscaras directamente con el ratón, o de mover el segmento de selección de línea, que no siendo estrictamente necesarias, mejoran la experiencia del usuario.
- **Gestión de listas:** Se han incluido todas las operaciones necesarias para gestionar cómodamente una lista, reordenar, editar elementos, eliminar lista completa... Además en la lista de imágenes se permite cambiar la etiqueta de cada imagen directamente sobre el elemento en la lista, con los métodos habituales del sistema operativo.
- **Barra de estado:** Tanto en la ventana principal, como en el modo “en vivo” existe una barra de estado que muestra al usuario información acerca del tipo de cálculo utilizado, el directorio de salida o el estado de la cámara.
- **Barra de título con asterisco de guardado:** El programa muestra en la barra de título la dirección del proyecto abierto. Además implementa el habitual asterisco al lado del nombre para indicar cuando el proyecto no está guardado.
- **Avisos de guardado:** Cada vez que el programa se va a cerrar, o se va a cargar otro proyecto, si el proyecto no está guardado, se muestra una alerta para recordar el guardado al usuario. Esta alerta redirigirá a “Guardar”, o a “Guardar Como” en función de si el proyecto ya había sido guardado.
- **Errores:** El programa implementa alertas de error para los errores más comunes, relacionados con la cámara, o la carga y guardado de archivos. Además se han realizado pruebas de forma exhaustiva para que el programa esté lo más libre de *bugs* posible.

- **Preferencias persistentes:** Guarda todas las preferencias de la cámara y del programa en una carpeta del sistema operativo para que se conserven para las futuras ejecuciones.
- **Instalador:** La aplicación se acompaña de un instalador/desinstalador, que ahorra al usuario la necesidad de conocer las ubicaciones de los archivos y de las entradas de registro para realizar una instalación/desinstalación manual. No obstante, en el manual de usuario se indican todas esas ubicaciones para una posible desinstalación manual.

## 6. Trabajos relacionados

En esta sección se describen brevemente dos trabajos relacionados con el tema acerca del que versa el proyecto.

### 6.1. *Caracterización y optimización de un jet de gas*

El primero de ellos es el Trabajo de Fin de Máster del cotutor Francisco Valle Brozas, titulado “Caracterización y optimización de un jet de gas para aceleración de electrones con láser”, presentado en Junio de 2012 [8].



Figura 105. Portada TFM Francisco Valle Brozas

Este trabajo, así como su autor, ha sido una referencia fundamental en la vertiente teórica del proyecto, permitiendo alcanzar los conocimientos necesarios en la materia para construir la aplicación.

Nótese que durante la realización del trabajo de fin de máster, la aplicación no estaba acabada, por lo que el procesado se tuvo que hacer de forma manual, imagen a imagen, con la aplicación que se expone en el siguiente apartado, junto con MATLAB®.

Este hecho fue el que permitió ver la gran utilidad que podría tener un programa especializado para ello, que permitiera hacerlo además a tiempo real, especialmente para los inminentes experimentos con aceleración de electrones.

## 6.2. IDEA: Interferometrical Data Evaluation Algorithms

Esta aplicación es la que se utilizaba con anterioridad a la realización del proyecto para el procesado de las imágenes del jet de gas.

Es una aplicación gratuita de ámbito científico para uso no comercial. No es de código abierto, por lo que no ha sido de gran ayuda en la generación del código, sin embargo su extenso manual acerca de los algoritmos que permite aplicar ha permitido mejorar la compresión de algunos de ellos, especialmente para la inversión de Abel[12].

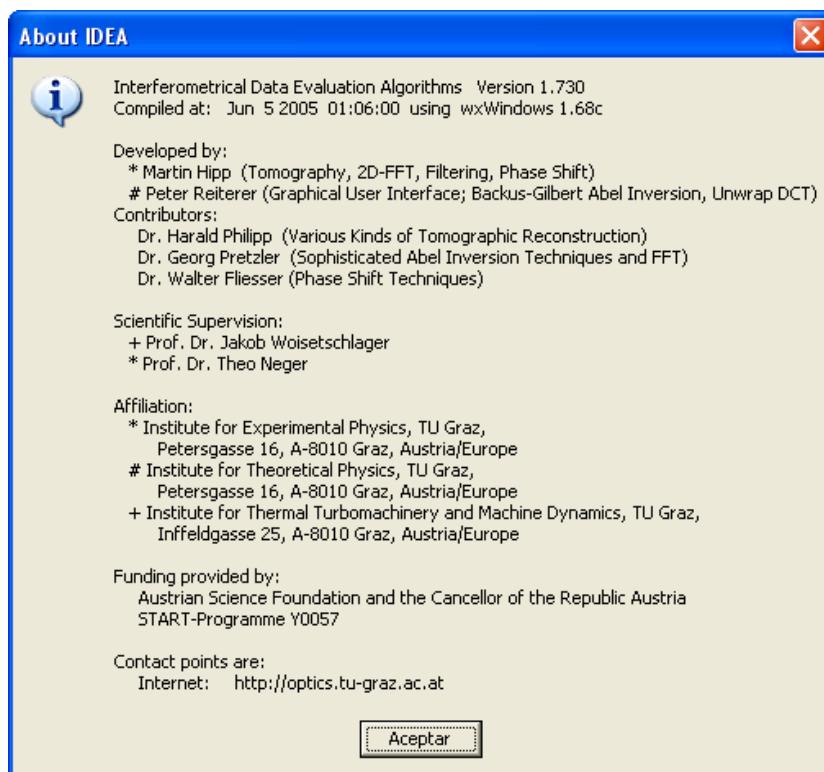


Figura 106. Acerca de IDEA

Aunque es una aplicación específica para procesar imágenes interferométricas, está pensada para su uso con imágenes aisladas, por lo que hace muy lento el trabajo con ella para obtener imágenes del jet de gas a partir de secuencias completas de

fotografías tomadas con la cámara, y hace totalmente impensable la visualización en directo de las imágenes.

Dado que es una aplicación de tratamiento de imágenes, usa el clásico sistema de ventanas dentro de la ventana principal, para cada imagen abierta, o transformada a partir de otra.

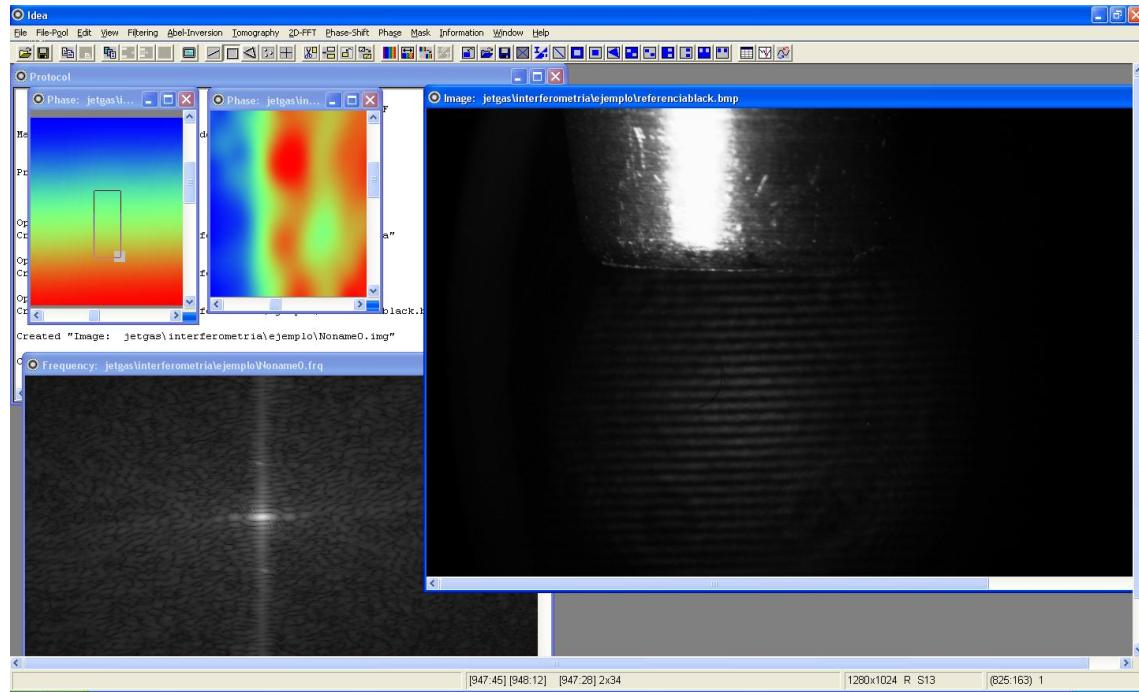


Figura 107. Imagen de la aplicación IDEA



## 7. Conclusiones

Tras la realización del proyecto se ha llegado a una serie de conclusiones en torno a diferentes aspectos:

- Respecto a la planificación del proyecto, se considera que ha sido un éxito. El hecho de tener desde el principio del proyecto unos plazos y objetivos realistas con fechas explícitas, junto a la fuerza de voluntad y el trabajo necesarios, ha permitido la realización de un proyecto más que satisfactorio para el autor con una duración admisible, sin extenderse demasiado en el tiempo, gracias a una focalización completa en el trabajo.
- Se ha experimentado la importancia de obtener y validar una serie de requisitos con el cliente de forma correcta.
- Se ha descubierto de forma práctica la importancia del análisis y el diseño según la Ingeniería del Software.

Es cierto que durante las asignaturas de la titulación, se adquieren estos conocimientos, pero hasta que no se aplican a un sistema real de una magnitud superior, como puede ser el proyecto de fin de carrera, no se aprecia la gran importancia que tiene la correcta documentación y generación de diagramas, que ayuden incluso al propio creador a refrescar de un vistazo el modo de funcionamiento de cierto módulo, o la sintaxis de salida de cierto archivo.

Por otra parte, se han descubierto las ventajas y el potencial de una programación orientada a objetos, con abstracción y herencia realizadas de forma correcta, y con uso de patrones en el diseño.

- Se han adquirido conocimientos para el uso de distintas bibliotecas y la propia biblioteca de Qt, así como en programación en paralelo, para la GPU. Especialmente este nuevo paradigma de programación ha resultado de gran agrado para el que escribe estas líneas y se espera que, dado su potencial, los conocimientos adquiridos sean útiles en el futuro.
- Respeto de la aplicación creada:

- Se ha logrado crear desde cero una aplicación completa, que realice unas tareas, especificadas según unos requisitos.
  - Para ello, se ha logrado poner el diseño sobre código, incluso sobre un lenguaje prácticamente desconocido antes de la realización del proyecto como es C++.
  - Se han superado todas las trabas encontradas a lo largo de la implementación relacionadas con la compatibilidad de las bibliotecas, o su compilación con Qt+Visual Studio.
  - Se ha utilizado una gran variedad de herramientas distintas para lograr el objetivo final, logrando una convivencia adecuada entre ellas, sin acoplamiento.
  - Se ha conseguido una correcta optimización del procesado de la imagen, alcanzando satisfactoriamente los objetivos de imágenes procesadas por segundo.
  - Además de cumplir los requisitos se ha creado una aplicación con una utilidad manifiesta en la vida real del laboratorio. Por ello se han cuidado algunos de los detalles de usabilidad que se pueden encontrar en una aplicación comercial como alertas de guardado, preferencias del programa, diferentes modos de exportado, instalador automático..., en definitiva, se ha logrado el objetivo de realizar una aplicación cómoda y práctica que se pueda usar sin apreciar carencias.
- Respecto de la documentación:
- Se ha logrado generar una documentación adecuada que incluso, sorprendentemente para el alumno, se ha llegado a disfrutar elaborando.
  - Se ha logrado separar los distintos puntos de vista a la hora de documentar una aplicación, la planificación, el análisis, el diseño, los manuales programador, los manuales de usuario, así como una memoria con un interesante apartado teórico, y una recopilación, a veces extendida, a veces reducida, de los aspectos más interesante de los otros apartados.

- Se ha aprendido a utilizar distintas herramientas para la generación de documentación, nunca antes utilizadas, como “Doxygen”.
- Se ha utilizado el programa en el laboratorio a tiempo real, para comprobar su funcionamiento, y se ha podido ver su potencial, respecto a las aplicaciones anteriormente usadas para tareas parecidas.



## 8. Líneas de trabajo futuras

Dado que el software trabaja en un contexto muy específico, las líneas de trabajo futuras, giran en torno al aumento de la compatibilidad con diferente hardware, diferentes experimentos...

### 8.1. Nuevas cámaras

El proyecto se ha implementado para funcionar con las cámaras del laboratorio, así como con el simulador de cámara.

A pesar de que las cámaras más usadas en investigación con láser son compatibles con la API µEye, existen otros tipos de cámara.

Una línea de futuro, sería aumentar el grado de compatibilidad con nuevos tipos de cámaras, y la inclusión de un nuevo cuadro de diálogo que permita seleccionar entre varios tipos de cámaras dentro del programa.

Nótese que llegado el momento, este proceso sería muy sencillo de realizar, dado que para el manejo de la cámara se creó una clase abstracta que definía una interfaz de métodos que cualquier cámara debería implementar, por lo que añadir nuevas cámaras se limita prácticamente a derivar nuevas clases de la clase abstracta, que sigan la especificación indicada en la documentación.

### 8.2. Nuevas formas de exportar resultados

Aunque en la aplicación ya se han proporcionado varias formas de exportar resultados, siempre pueden ser necesarios nuevos formatos para los archivos de salida.

La dificultad para realizar esto dependería, obviamente, del formato nuevo para el archivo exportado, pero en lo que al programa respecta, bastaría con añadir un nuevo elemento de menú, y hacer las peticiones a los métodos ya existentes de los controladores para obtener los datos a partir de los que se generara la salida.

### ***8.3. Nuevos algoritmos***

Como ya se ha comentado, la aplicación creada sirve, no solamente para llegar a obtener la imagen de un jet de gas, sino de cualquier medio, objeto, o sustancia que pueda ser atravesada por un láser, y que introduzca retardos pequeños de fase.

Sin embargo, los algoritmos incluidos para el procesado de la imagen, una vez obtenida la primera imagen después del pre-procesado, están más pensados para el tratamiento de algo cilíndricamente simétrico.

Por lo tanto una de las posibles líneas de trabajo futuras consiste en añadir nuevos algoritmos que permitan realizar otras operaciones deseadas, para mejorar la imagen conseguida.

No obstante, este trabajo no sería difícil de realizar, y, a mayores de implementar los algoritmos en las clases correspondientes, sólo habría que modificar una mínima parte de código fuente.

### ***8.4. Nuevos modos de cálculo***

Aunque CUDA es una herramienta que satisface las necesidades actuales, existen otras alternativas para el cálculo que se podrían añadir a la aplicación sin más que implementar las clases adecuadas para el sistema de herencia y factorías ya proporcionado.

Una de estas alternativas es OpenCL. OpenCL es una herramienta libre para computación con la GPU que, al igual que pasó con OpenGL para la programación de aplicaciones 3D, tiene muchas posibilidades de convertirse en el estándar más usado en el campo.

Para la elaboración del proyecto se eligió CUDA respecto a OpenCL principalmente por dos motivos:

- Es un sistema mucho más maduro que se encuentra en su versión 4.2, frente a la versión 1.2 de OpenCL.
- Dado que OpenCL es compatible con más tarjetas de distintos fabricantes, entre ellos NVIDIA, no está tan optimizado por ahora para las arquitecturas de cada tarjeta, por lo que es algo menos eficiente, y generalmente más lento.

No obstante es probable que esas dos cuestiones se resuelvan o atenúen con el tiempo, pudiendo llegar a hacer a OpenCL una alternativa más interesante que CUDA.



## 9. Agradecimientos

Y no se quiere finalizar este proyecto sin agradecer a todas las personas implicadas, directa o indirectamente con su realización, sin las que este proyecto no hubiera sido posible.

En primer lugar a Guillermo González Talaván, por su labor en su rol de tutor, estando disponible cuando ha sido necesario, aportando ideas para el proyecto y revisando el correcto transcurso de todo el proyecto, mostrando siempre competencia a la hora de resolver dudas técnicas para lograr los objetivos fijados.

En segundo lugar a Francisco Valle Brozas, cotutor del proyecto y cliente de la aplicación, por la concepción inicial del proyecto, todo el tiempo invertido en la toma de requisitos y la validación de todos los resultados que se iban generando, sin olvidar también toda la ayuda teórica sobre el tema que ha proporcionado, la revisión de la memoria y las tardes pasadas alineando el sistema y probando la aplicación en el laboratorio.

También se quiere agradecer a Isabel Arias Tobalina, profesora del Departamento de Física Aplicada por toda su ayuda a la hora de conseguir el material y los medios necesarios para poder disponer de un despacho y un ordenador compatible con CUDA en el edificio Trilingüe para poder trabajar más cerca del equipo experimental, así como por su interés a lo largo del transcurso de todo el proyecto, incluyendo una revisión de la parte teórica de la memoria.

Y por último, pero no por ello menos importante, a mi familia, novia y amigos, acompañándome de cerca, especialmente en los momentos difíciles, de frustración y cansancio, animando siempre a seguir adelante, y gracias a los que he sacado fuerzas para intentar dar lo mejor de mí día tras día, hasta el final.



## 10. Referencias y bibliografía

- [1] Grupo de Óptica del departamento de Física Aplicada de la Universidad de Salamanca. [Online]. <http://optica.usal.es/>
- [2] Departamento de Física Aplicada de la Universidad de Salamanca. [Online]. <http://campus.usal.es/~fisapli/>
- [3] Universidad de Salamanca. [Online]. <http://www.usal.es>
- [4] IDEA (Interferometric Data Evaluation Algorithms). [Online]. <http://www.optics.tugraz.at/idea/idea.html>
- [5] P. Tipler and G. Mosca, *Physics for Scientists and Engineers with Modern Physics*, Sixth Edition ed.: W.H. Freeman & Company, 2007.
- [6] Rocío Borrego, Carlos Hernández García, Carolina Romero, and José Antonio Pérez, *El láser, la luz de nuestro tiempo*, Primera Edición ed.: B. Alonso, 2010.
- [7] Wikipedia: The Free Encyclopedia. [Online]. <http://en.wikipedia.org>
- [8] Francisco Valle Brozas, "Caracterización y optimización de un jet de gas para aceleración de electrones con laser," Universidad de Salamanca, 2012.
- [9] *Numerical Recipes: The Art of Scientific Computing*, Third Edition ed.: Cambridge University Press., 2007.
- [10] *Schaum's Outline of Mathematical Handbook of Formulas and Tables*, Third Edition ed.: Schaum Outline Series, 2008.
- [11] J. Wolberg, *Data Analysis Using the Method of Least Squares: Extracting the Most Information from Experiments.*: Springer, 2005.

- [12] Manual de IDEA. [Online]. <http://www.optics.tugraz.at/idea/manual.html>
- [13] Página web de Qt. [Online]. <http://qt.nokia.com/>
- [14] Web de Visual C++ 2008 Express Edition. [Online].  
<http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express>
- [15] Web de QwtPlot. [Online]. <http://qwt.sourceforge.net/>
- [16] Web de QwtPlot3D. [Online]. <http://qwtplot3d.sourceforge.net/>
- [17] Web de QuaZIP. [Online]. <http://quazip.sourceforge.net/>
- [18] Manual de la API uEye de IDS-Imaging. [Online]. [http://www.ids-imaging.de/frontend/files/uEyeManuals/Manual\\_eng/uEye\\_Manual/](http://www.ids-imaging.de/frontend/files/uEyeManuals/Manual_eng/uEye_Manual/)
- [19] Página web oficial de CUDA por NVIDIA. [Online].  
<http://developer.nvidia.com/category/zone/cuda-zone>
- [20] Tarjetas de NVIDIA compatibles con CUDA. [Online].  
<http://developer.nvidia.com/cuda/cuda-gpus>
- [21] Página de descarga del kit de desarrollo para CUDA. [Online].  
<http://developer.nvidia.com/cuda/cuda-downloads>
- [22] Jason Sanders and Edward Kandrot, *CUDA by Example*, First Edition ed.: Addison-Wesley, 2010.
- [23] North Carolina State University Media Wiki. [Online]. [http://pg-server.csc.ncsu.edu/mediawiki/index.php/CSC\\_ECE\\_506\\_Spring\\_2011/ch2a\\_mc](http://pg-server.csc.ncsu.edu/mediawiki/index.php/CSC_ECE_506_Spring_2011/ch2a_mc)
- [24] Understanding the CUDA Data Parallel Threading Model. [Online].  
<http://www.pgroup.com/lit/articles/insider/v2n1a5.htm>

[25] Web de MPlayer. [Online]. <http://www.mplayerhq.hu>

[26] Web de Image Magick. [Online]. <http://www.imagemagick.org>

[27] Web de Real World. [Online]. <http://www.rw-designer.com>

[28] Web de ClickTeam. [Online]. <http://www.clickteam.com>

[29] Web de Microsoft Office. [Online]. <http://office.microsoft.com>

[30] Web de Doxygen. [Online]. <http://www.doxygen.org>

[31] Web de REM. [Online].

[http://www.lsi.us.es/descargas/descarga\\_programas.php?id=3](http://www.lsi.us.es/descargas/descarga_programas.php?id=3)

[32] Web de Visual Paradigm for UML 8.0. [Online]. <http://www.visual-paradigm.com/>

[33] Web de Adobe Photoshop. [Online].

<http://www.adobe.com/es/products/photoshop.html>