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

Resumen de la Memoria

Proyecto de Fin de Carrera
INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Septiembre de 2012

Autor
Álvaro Sánchez González

Tutor
Guillermo González Talaván

Cotutor
Francisco Valle Brozas

Contenido

Índice de Figuras

1. Introducción

El proyecto surge en el entorno del Área de Óptica del Departamento de Física Aplicada de la Universidad de Salamanca, motivado por la necesidad de un software que permitiera y agilizara la caracterización de un jet de gas a tiempo real para la aceleración de electrones con láser.

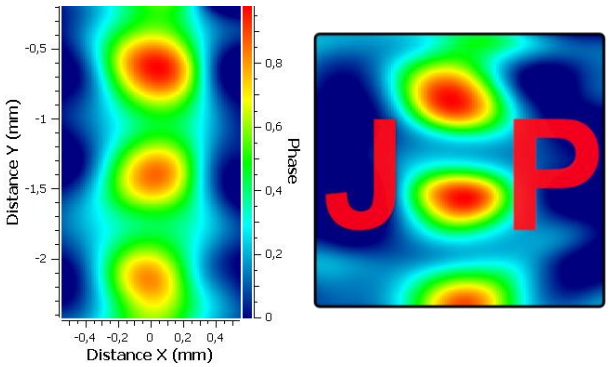


Figura 1. Imagen de un jet de gas y logo de JetProcessing

Un "jet de gas" consiste en una corriente de gas generada artificialmente por un pistón y una válvula que controle el flujo, y "caracterizar un jet de gas" consiste en determinar la densidad o índice de refracción del gas en cada punto del espacio.

Realizar esa tarea no es fácil ya que, por una parte, el jet de gas es transparente, por lo que a simple vista es imposible de observar, y por otra parte, si se introdujera cualquier tipo de sonda, se modificaría la estructura del jet, afectando a la medida.

La solución a esto es el uso de un sistema de interferometría con láser, que permita, mediante el uso de un láser pulsado, la generación de un patrón de interferencia tras cuyo pre-procesado y procesado se pueda obtener lo que se busca.

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, diez por segundo, lo que, en la práctica, resulta prácticamente imposible para una CPU normal.

Por ello, para aumentar el rendimiento de la aplicación, se ha utilizado la herramienta CUDA de NVIDIA que permite procesar las imágenes en paralelo con la tarjeta gráfica.

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", permite realizar todas las operaciones habitualmente necesarias para la caracterización de no sólo un jet de gas, sino de cualquier material o substancia transparente candidata de ser caracterizada en un sistema de interferometría.

1.1. Objetivos del proyecto

Los objetivos 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.
- **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.
- **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 para poder llegar a mostrar el índice de refracción del gas.
- **Tiempo real:** la aplicación deberá poder realizar y mostrar los cálculos a la mayor velocidad posible y a tiempo real, según va tomando las imágenes de interferencia sobre el sistema. Idealmente debería procesar hasta 10 imágenes por segundo. Para ello se utilizará la herramienta CUDA de NVIDIA que permitirá realizar los cálculos necesarios con la GPU 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 en texto plano.
- **Gestión de la cámara:** el sistema deberá poder configurar los parámetros habituales de una cámara y utilizarla para tomar las imágenes.
- **Gestión de proyectos:** el sistema deberá ser capaz de tratar con proyectos completos, para que el usuario investigador pueda retomar el trabajo por donde lo dejó el día anterior de forma efectiva.
- **Usabilidad del programa:** la aplicación final 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.
- **Mayor compatibilidad posible:** Aunque inicialmente la aplicación sólo se construirá para sistemas Windows, se usarán siempre bibliotecas y herramientas multiplataforma para facilitar la posible migración.

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 imágenes de interferencia, 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 incluso generar gráficos adecuados para su uso en pósters o publicaciones, además de poder obtener los resultados en formato de texto plano, que permitan su posterior uso en aplicaciones como MATLAB® o Wolfram Mathematica®.

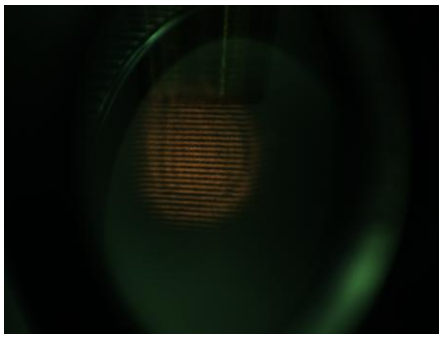
2. Técnicas y herramientas

Para lograr desarrollar la aplicación ha sido necesaria la utilización de distintas herramientas, todas ellas multiplataforma, que se resumen a continuación:

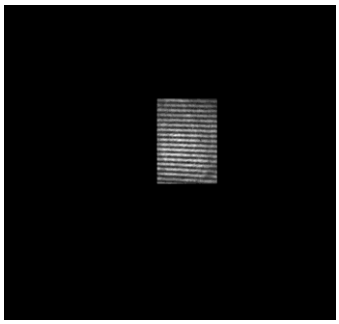
- **Programación orientada a objetos con C++.**
- **Biblioteca Qt y Qt Creator:** Para la programación de la interfaz gráfica y toda la interacción con el sistema operativo, como la creación de hilos o acceso a carpetas del sistema, permitiendo así que todo ello sea multiplataforma.
- **CUDA:** La herramienta de NVIDIA para el procesamiento en paralelo de datos, creando código con una paralelización adecuada para su ejecución en la GPU.
- **Compilador de Visual C++ Express Edition:** Por compatibilidad de la herramienta CUDA bajo sistemas Windows. No necesario en otras plataformas.
- **Bibliotecas de representación gráfica QwtPlot y QwtPlot3D:** Para la representación de resultados, en dos y en tres dimensiones respectivamente.
- **Biblioteca de compresión QuaZIP:** Biblioteca de tratamiento con archivos comprimidos ZIP para la creación de los archivos de proyecto.
- **API µEye:** Para la programación de las cámaras del laboratorio.
- **Archivos INI:** Para el guardado de parámetros sencillos de forma persistente.
- **Archivos XML:** Para el guardado de estructuras de datos más complejas de forma persistente.
- **MPlayer:** Con su herramienta MEncoder, para la exportación de resultados en formato de vídeo AVI.
- **ImageMagick:** Con su herramienta Converter, para la exportación de animaciones en formato GIF.
- **RealWorld Icon Editor:** Para la edición de los iconos utilizados en el programa.
- **Click Team Install Creator Pro:** Para la creación del instalador.
- **Herramientas para la documentación:** Microsoft Word 2007, Doxygen, REM 1.2.2, Visual Paradigm for UML 8.0, Microsoft Paint, Microsoft Excel 2007 y Adobe Photoshop CS.

3. Etapas de procesado de una imagen de interferencia

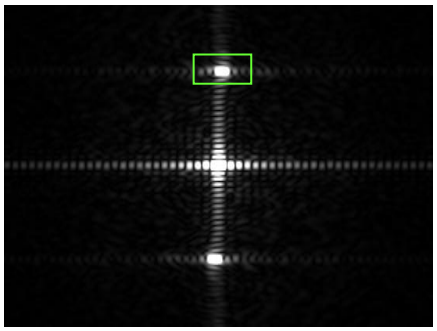
El procesamiento completo de una imagen de interferencia comienza con el pre-procesado junto con una imagen de referencia, realizando para ambas los siguientes pasos:



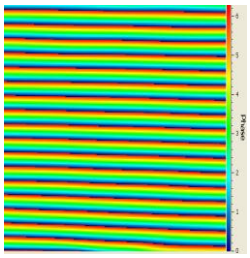
- Obtener la imagen inicial, o bien de un archivo, o bien directamente de la cámara.



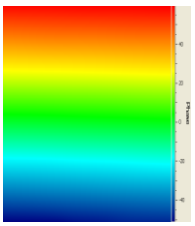
- Establecer unas dimensiones adecuadas para la imagen, de forma que sean potencia de 2.
- Establecer una máscara dejando la zona de interés de la imagen.



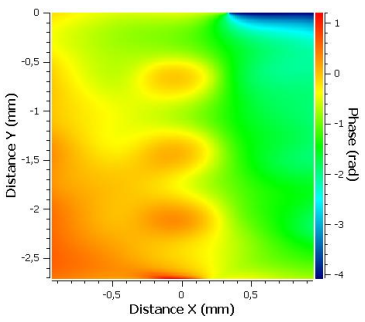
- Realizar la transformada de Fourier.
- Aplicar otra máscara para mantener sólo el término de la fase.



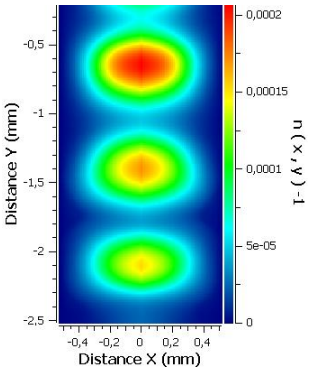
- Realizar la transformada de Fourier Inversa.
- Obtener la fase en la zona de interés.
- Hacer continua la fase, eliminando los saltos entre 2π y 0.



Ahora, restando las fases obtenidas para la imagen con el jet de gas presente y la imagen de referencia, se obtiene una primera imagen del jet de gas.



Esa primera imagen se procesa posteriormente con diferentes operaciones de eliminación de ruido y cálculo numérico, para llegar, por ejemplo, al índice de refracción.



4. Aspectos relevantes del desarrollo del proyecto

Esta sección incluye una selección de los aspectos más interesantes del desarrollo.

4.1. Ciclo de vida utilizado

Para el desarrollo del proyecto se fijaron desde el principio una serie de plazos explícitos con sus correspondientes hitos, lo que facilitó sobremanera la distribución del trabajo.

No obstante, los plazos marcados se ajustan perfectamente al Proceso Unificado, con una serie de iteraciones distribuidas en las siguientes etapas:

- **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 mediante 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. Consistió en 3 iteraciones: diseño del sistema base, diseño de la interfaz y refinamiento del sistema base.
- **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. Constó de 4 iteraciones principales: búsqueda del entorno de desarrollo, implementación del sistema base, puesta de los elementos en conjunto e implementación de la funcionalidad añadida, conteniendo cada una de ellas una gran cantidad de pequeñas iteraciones de carácter incremental.
- **Fase de transición (01/08/2012-15/09/2012**): Al comenzar esta fase se cuenta con una versión beta del proyecto sobre la que realizar pruebas y pequeñas modificaciones. Además, en paralelo, se redacta la memoria definitiva, recopilando para los anexos todos los artefactos que ya se habían generado.

4.2. Patrón Modelo-Vista-Controlador modificado

Para la gestión de la interfaz gráfica se ha utilizado un patrón Modelo-Vista-Controlador modificado, que reduce al mínimo las dependencias entre los componentes. Esta versión del patrón distribuye las tareas de la siguiente forma:

- Vista: Es una clase generada automáticamente por el diseñador de Qt. Contiene todos los visualizadores y controles y se encarga de enviar señales cada vez que se produzca una acción por parte del usuario.
- Controlador: Existe uno para cada vista. Se encarga de recibir todas las señales procedentes de la vista y en base a ellas modificar el modelo o la vista según proceda.
- Modelo: Contiene la lógica de cálculo y los datos de la aplicación. Recibe peticiones del controlador. En la aplicación creada, el modelo contiene a su vez otro controlador, no ya relacionado con la interfaz, sino con la organización del programa, que da acceso al resto de los elementos.

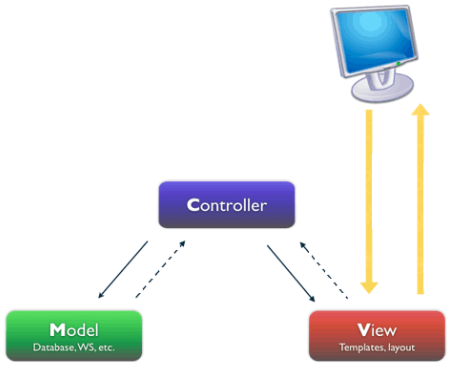


Figura 2. Patrón MVC modificado

Siguiendo este esquema, y gracias al sistema señal/*slot* de Qt, las únicas dependencias explícitas son del controlador a la vista y del controlador al modelo.

4.3. Breve descripción de la interfaz gráfica

La aplicación final cuenta con dos clases de almacenamiento de resultados: Data1D y Data2D, que guardan información de fase a lo largo de una línea o de una superficie rectangular, respectivamente.

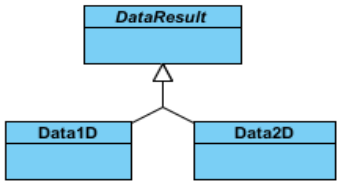


Figura 3. Clases de resultados

Para mostrar estos resultados se han creado 4 clases gráficas distintas, como nodos hoja en una jerarquía de herencia, una de ellas para objetos del tipo Data1D y otras tres para objetos del tipo Data2D.

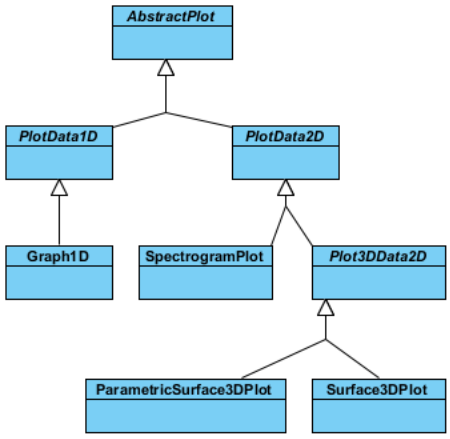


Figura 4. Jerarquía de gráficos

Estas clases gráficas tienen la peculiaridad de contar con varias opciones de escalado y de ser totalmente interactivas para el usuario permitiendo la representación de largas secuencias de resultados manteniendo las operaciones realizadas sobre el gráfico.

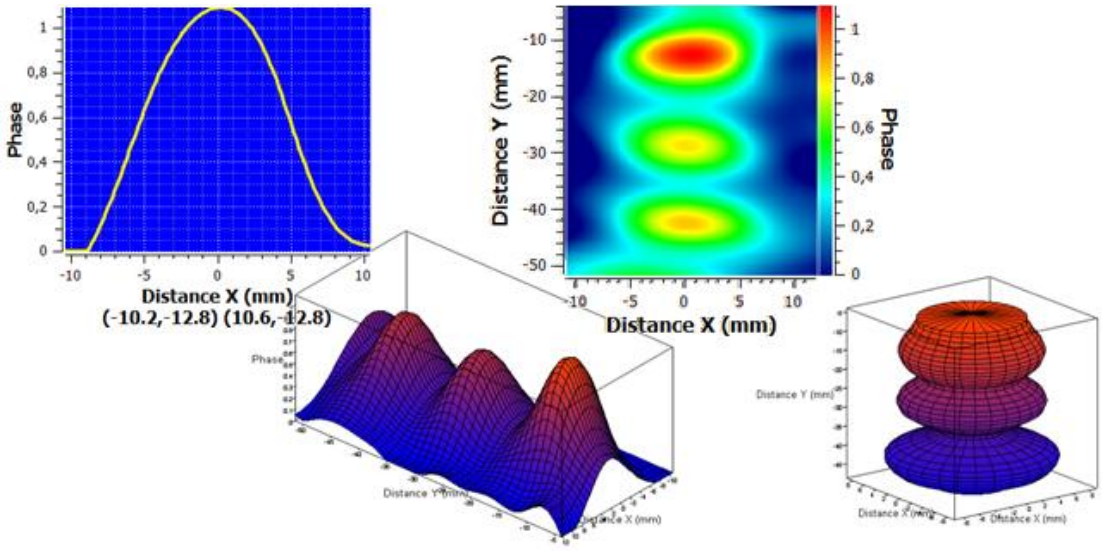
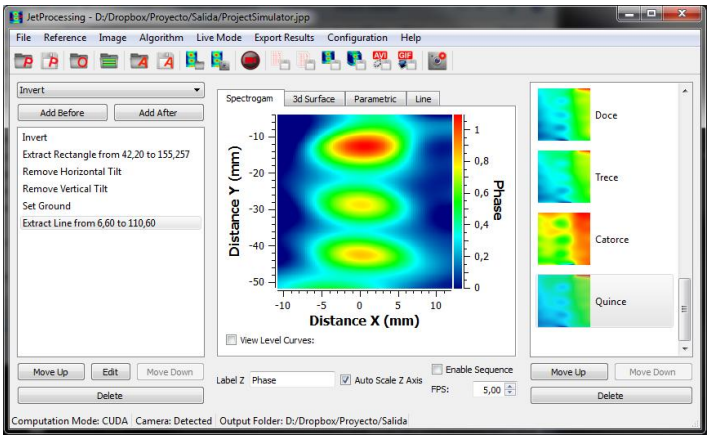


Figura 5. Tipos de gráficos

Además hay más de 10 ventanas y cuadros de diálogo disponibles para acceder a la distinta funcionalidad de la aplicación, destacando los que se pueden apreciar en la Figura 6.



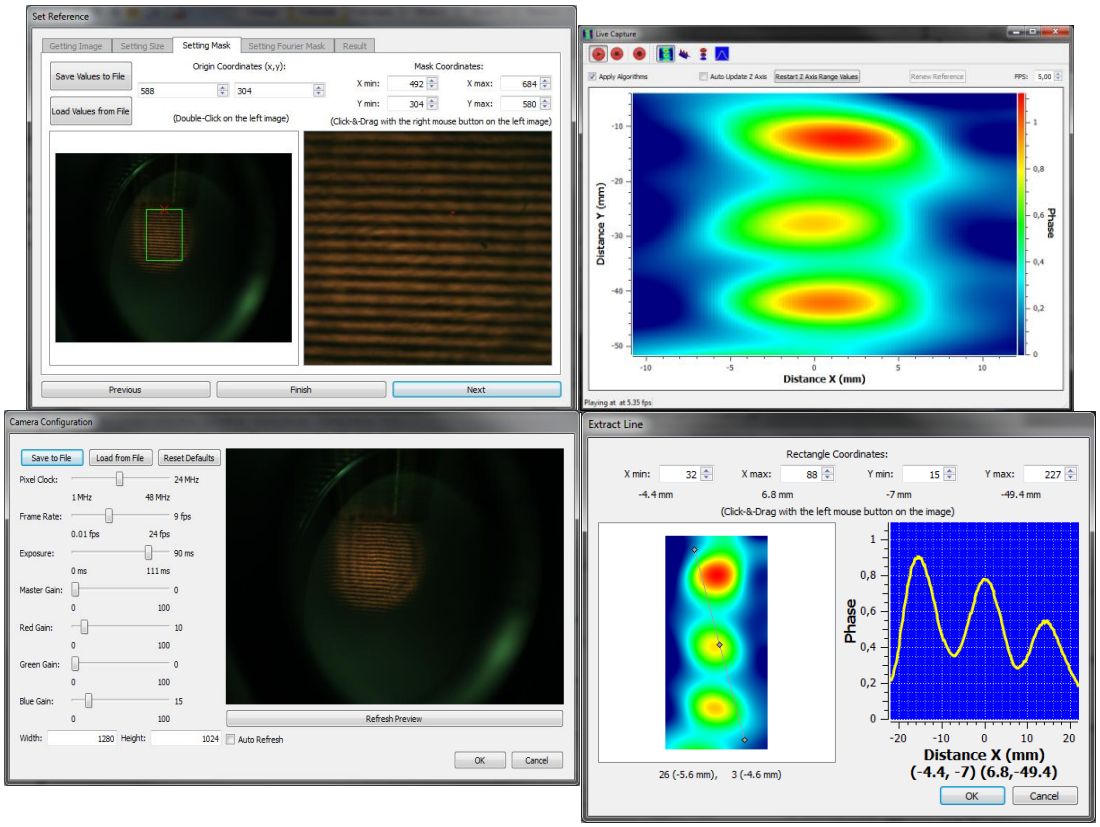


Figura 6. Algunas ventanas y diálogos

4.4. Estructura estática del sistema base

La vista estática a primer nivel del sistema base cuenta con tres controladores completamente independientes: el controlador de imágenes, de algoritmos y de cámaras, organizados a su vez por un controlador principal.

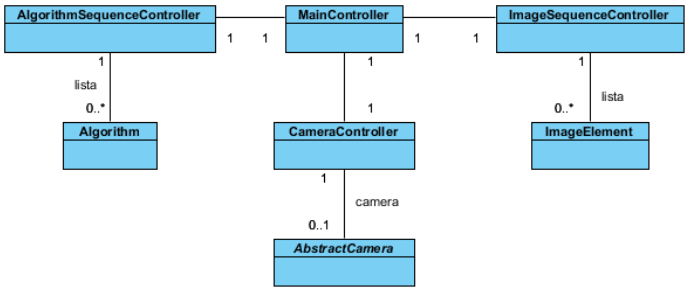


Figura 7. Vista estática a primer nivel

Cada uno de esos cuatro controladores tiene además una serie de relaciones a segundo y tercer nivel, con distintas clases pertenecientes a módulos independientes entre sí, organizados en capas, y entre los que se distribuyen las más de 40 clases existentes en el sistema base, sin contar las relacionadas con la interfaz gráfica.

4.5. Abstracción y uso de factorías

Una de las claves en el diseño del proyecto ha sido la utilización de abstracción y de factorías.

Esta estrategia se ha utilizado en dos ocasiones:

- Uno de los objetivos era que los cálculos se pudieran realizar con la GPU mediante la herramienta CUDA de NVIDIA, permitiendo también opcionalmente los cálculos con la CPU. Para ello se definieron unas clases abstractas con una interfaz de métodos que contuvieran los datos sobre los que se realizaban los cálculos, de forma que, para realizar una implementación concreta de los métodos, se heredara de la clase abstracta.

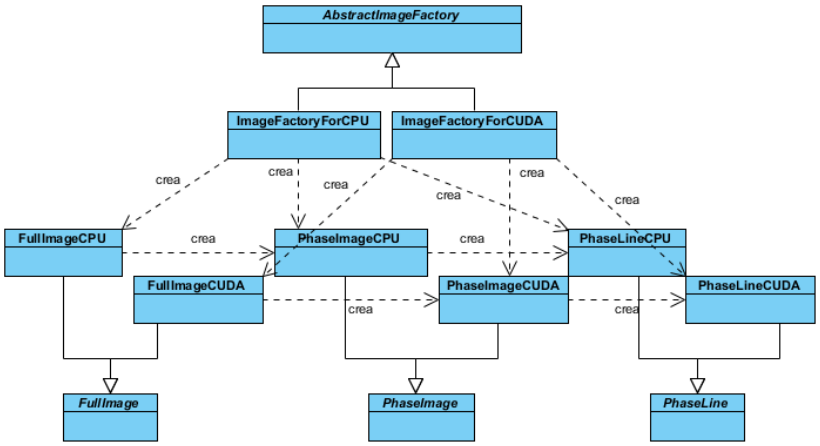


Figura 8. Uso de factorías para los cálculos

- De la misma forma, para la gestión de la cámara, se especificó una clase abstracta definiendo las operaciones más comunes realizables sobre este tipo de dispositivos, de la que se heredara para implementar cámaras concretas. En este caso se utiliza para implementar una cámara µEye y un simulador de cámara que carga imágenes desde archivo.

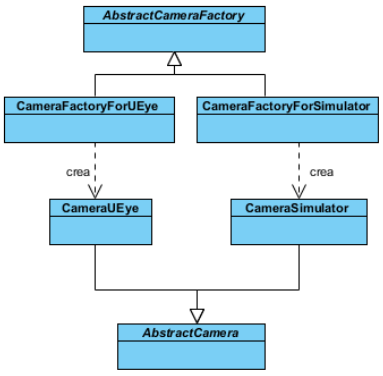


Figura 9. Factorías para la cámara

Además, en ambos casos, se utilizaron factorías para la creación/copia de los objetos de las clases derivadas como si de elementos abstractos de trataran, logrando así limitar a una sola zona del código la elección entre los tipos de clase derivada.

4.6. Realización de cálculos en paralelo

Para aumentar el rendimiento a la hora de realizar cálculos se utiliza, como ya se comentó, la herramienta CUDA de NVIDIA. Para ello ha sido necesario establecer una paralelización para todos los cálculos que se realizan, optimizando de forma explícita la distribución del trabajo en bloques e hilos, para su ejecución en los multiprocesadores de las tarjetas gráficas compatibles.

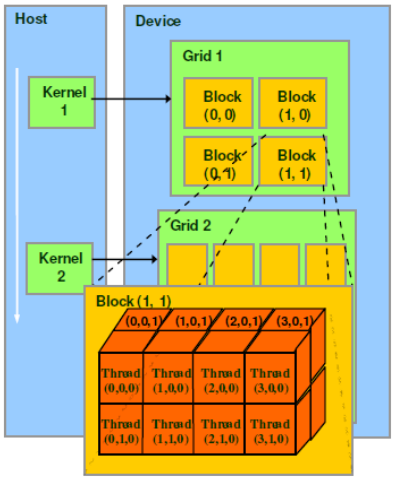


Figura 10. Distribución del procesamiento con CUDA

4.7. Toma de imágenes de forma eficiente

Uno de los objetivos de la aplicación consiste en realizar todo el proceso de obtención de imagen del jet de gas a la mayor velocidad posible. Hasta ahora se ha hablado de la parte del procesado la imagen, usando procesamiento en paralelo con CUDA 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 un tiempo de exposición de unos 100 ms, durante los cuales el ordenador 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.

Para evitar esa desocupación de CPU/GPU, que se podría utilizar para procesamiento de imagen, se ha utilizado la siguiente estrategia, basada en hilos, y gestionada por el controlador de la cámara:

- Al iniciar el controlador de la cámara, éste creará un hilo en paralelo, que tomará una fotografía, y la almacenará en un buffer.

- Cuando en alguna parte del programa se necesite una imagen, se solicitará al controlador, que devolverá la imagen actualmente en el buffer.
- Acto seguido, el hilo en paralelo comenzará a tomar otra imagen para el buffer.

De esta forma, para el procesamiento continuo de imagen, se puede llegar a reducir el tiempo total de obtención de la imagen final del jet al mayor de los dos: el tiempo en tomar la imagen o el tiempo en procesar la imagen, en vez de a la suma de ambos.

Todo este proceso, que tiene añadidos como la modificación de parámetros simultáneamente con la toma de imágenes, o el renovado de la imagen del buffer para asegurar que esté actualizada cuando esto pueda ser necesario, se sincroniza mediante tres semáforos y un *mutex*.

4.8. 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 que se podrían procesar y mostrar con un solo hilo. No obstante, en esta aplicación, para la que se busca una frecuencia de procesamiento lo más alta posible y cuyos gráficos son interactivos, esa forma de trabajar no es viable. El motivo es que la mayor parte del tiempo tendría el control la parte de procesamiento, mientras que la interfaz quedaría bloqueada recibiendo el control durante sólo unos pocos mili-segundos cada segundo. Esto haría imposible responder a las acciones del usuario de forma agradable para la experiencia de éste, como por ejemplo al girar un gráfico 3D.

La forma de evitar ese problema es, de nuevo, recurrir a los hilos, de modo que, cada vez que el controlador de la interfaz detecte que tiene que obtener y mostrar una nueva imagen del jet de gas, se cree una instancia de la clase DataProducer en otro hilo. Esta instancia irá procesando una imagen de la cámara en paralelo al hilo que controla la interfaz, permitiendo mientras tanto al usuario interaccionar con dicha interfaz. Al final del procesamiento el hilo de cálculo avisará de nuevo al controlador de la interfaz, que tomará brevemente el control y representará los datos.

Además se incluye un semáforo que evita el procesado de más de una imagen a la vez.

4.9. Guardado en ficheros

Existen hasta 10 ficheros distintos de guardado de elementos del programa: seis de ellos en formato INI, dos en formato XML, uno binario y un archivo en formato ZIP.

Dos de los archivos INI se guardan en una carpeta de configuración local para conservar de forma persistente la configuración del programa y de la cámara.

El archivo en formato ZIP con extensión ".jpp" contiene ocho archivos de los anteriores, con los distintos elementos que conforman un proyecto, añadiendo un archivo de imagen con la referencia.

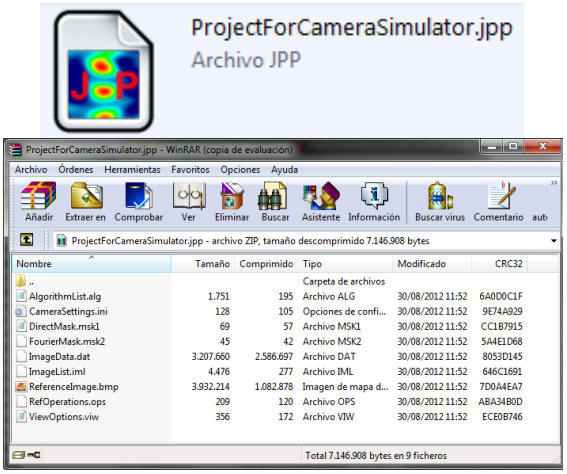


Figura 11. Archivo de proyecto

Además, los resultados se pueden exportar para una imagen concreta en formatos ".bmp", ".jpg" y ".png" o en formato de texto con la información de cada punto, así como para toda una serie de imágenes en los formatos anteriores, o incluso como una secuencia contenida en un vídeo en formato AVI o una animación en formato GIF.

4.10. Detalles para un uso más cómodo

En una aplicación es tan importante el hecho de que realice unas tareas de forma correcta, como el hecho de que proporcione unas facilidades de usabilidad que faciliten el trabajo en el día a día.

Para ello se han incluido una serie de características como los archivos de proyecto, asociación de su extensión con la aplicación, alertas de guardado, barra de título con nombre y estado del proyecto, barra de estado, existencia de un instalador, preferencias persistentes, múltiples formas de exportado, gráficos interactivos, manejo por ratón o teclado, guardado de parámetros en archivos separados, etc. que hacen que el trabajo con JetProcessing sea cómodo y llevadero para el usuario.

5. Conclusiones

Tras la realización del proyecto se ha llegado a una serie de conclusiones:

- 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, ha permitido la realización de un proyecto más que satisfactorio para el autor sin extenderse demasiado en el tiempo.
- 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, y especialmente, las ventajas y el potencial de una programación orientada a objetos, con abstracción y herencia realizadas de forma correcta, y con uso de patrones.
- Se han adquirido conocimientos de carácter general útiles para el futuro.
- Respecto de la aplicación creada:
    - Se ha logrado crear desde cero una aplicación completa, que realice unas tareas, especificadas según unos requisitos.
    - Se han superado todas las trabas encontradas a lo largo de la implementación relacionadas con la compatibilidad de las bibliotecas, o su compilación y enlazado con Qt+Visual Studio+CUDA.
    - Se han utilizado una gran variedad herramientas distintas para lograr el objetivo final, logrando una convivencia adecuada entre ellas.
    - Se ha conseguido una correcta optimización del procesado de la imagen, alcanzando satisfactoriamente los objetivos de rendimiento.
    - Además de cumplir los requisitos, se ha creado una aplicación con una utilidad manifiesta en la vida real del laboratorio, que se ha llegado a probar con resultado más que satisfactorio, y con una usabilidad muy cuidada.
    - Se ha generado una documentación completa, especificando todos los detalles desde distintas vertientes.

6. 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 para diferente hardware, diferentes experimentos…, y se pueden resumir en las siguientes:

- **Nuevas cámaras:** implementación de código para tipos de cámaras adicionales, así como una interfaz que permita elegir entre las distintas cámaras.
- **Nuevas formas de exportar resultados:** aunque ya se han proporcionado diversos formatos de salida para los resultados, siempre puede ser necesario alguno más según las exigencias de los usuarios.
- **Nuevos algoritmos:** las operaciones actualmente implementadas están optimizadas para el procesado de una imagen de un objeto con simetría cilíndrica. El aumento de la compatibilidad para otros tipos de muestras se lograría simplemente aumentando el número de operaciones disponibles para añadir a la lista de algoritmos.
- **Nuevos modos de cálculo:** nuevas formas de realizar las operaciones, alternativas a la CPU o a CUDA, como puede ser OpenCL.

Nótese que la aplicación actual ya contiene todas las estructuras suficientes para soportar la introducción de todos esos cambios, de forma que el trabajo sería mínimo, teniendo que modificar sólo algunas partes muy concretas del código en cada uno de los casos.