Curso 2011/12
CIENCIAS Y TECNOLOGÍAS/23
I.S.B.N.: 978-84-15910-20-6

JOSÉ GIL MARICHAL HERNÁNDEZ

Obtención de información tridimensional de una escena a partir de sensores plenópticos usando procesamiento de señales con *hardware* gráfico

Directores
FERNANDO LUIS ROSA GONZÁLEZ
JOSÉ MANUEL RODRÍGUEZ RAMOS

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

Più nessuno mi porterà nel Sud
Salvatore Quasimodo

Prefacio

Inicié mis estudios de doctorado en el programa Física e Informática en el curso 2003-04, poco después de haber finalizado la Ingeniería en Informática, bajo la dirección de los doctores Fernando Rosa y José M. Rodríguez Ramos.

Ambos directores son doctores en Física, especializados en Astrofísica, concretamente en alta resolución espacial y en óptica adaptativa, adscritos a las áreas de Teoría de la Señal y Comunicaciones; y Tecnología Electrónica, respectivamente. En el momento de la incorporación a su grupo de investigación se contaba con financiación pública para la investigación en torno a la adaptación e integración de sensores de frente de onda para la obtención de distancias. Dadas las características del proyecto abierto, la especialización de uno y otro director y del doctorando, las investigaciones en los primeros años giran en torno a la aplicación de las GPUs al tratamiento digital de las señales provenientes de un sensor de frente de onda. Con idea de, una vez se concretase el algoritmo más conveniente, aplicar sobre él los conocimientos desarrollados en la GPU, y en especial en el aprovechamiento de la velocidad en el cómputo de la FFT.

Al finalizar las fases docente e investigadora, sin embargo, redirigimos la investigación en pos del algoritmo de obtención de información geométrica, lo que se plasmó en el proyecto de tesis de fecha Septiembre de 2005. Prácticamente en la misma fecha Ren Ng propone la *fotografía por Slice de Fourier*. Donde se confirman las intuiciones en torno a las cuales se venía trabajando, pero que, como es evidente ya no pueden constituir un tema de Tesis Doctoral. Por fortuna, y pese a merecer en 2006 la distinción a la mejor Tesis Doctoral concedida por la *«Association for Computing Machinery»*, el trabajo de Ren Ng aún es susceptible de mejora. Por un lado desde el punto de vista teórico, algorítmico, y también desde un punto de vista práctico, de la implementación de la FFT en la que se basa.

El trabajo recogido en la presente memoria se ha desarrollado principalmente desde esa fecha hasta principios de 2008. Postergándose su publicación con el fin de patentar los principales resultados a los que dio lugar. Véanse al efecto las patentes [Rodríguez-Ramos 07] y [Rodríguez-Ramos 09].

Agradecimientos

Quiero hacer constar mi gratitud a mis directores de Tesis, al resto de compañeros del Departamento de Física Fundamental y Experimental, Electrónica y Sistemas de la Universidad de La Laguna, así como a miembros del Instituto de Astrofísica de Canarias con los que he colaborado, y a todas aquellas personas que de una u otra forma me han ayudado.

También quiero agradecer a CajaCanarias las becas recibidas en los primeros años de realización de mis estudios de doctorado.

Mención especial merecen mis padres, hermanos y demás familiares.

Pero este trabajo está dedicado a dos personas muy especiales: mi mujer, Marisol, y mi hijo, Daniel. Nadie me llevará al Sur. Ellos me lo trajeron.

José Gil Marichal Hernández

Índice general

1. Introducción **1**
1.1. Visión de conjunto . . . 3
1.1.1. El sensor . . . 4
1.1.2. El procesado . . . 5
1.1.3. El recurso computacional . . . 8
1.2. Hipótesis . . . 10
1.3. Objetivos . . . 10
1.3.1. Ventajas del enfoque adoptado . . . 11
1.3.2. Aplicaciones y objetivos secundarios . . . 11
1.4. Organización de esta memoria . . . 12

2. Marco teórico **13**
2.1. Fotografía computacional . . . 13
2.1.1. Función plenóptica . . . 17
2.1.2. Muestreo y sensores plenópticos . . . 24
2.2. Volumen focal . . . 30
2.2.1. Reenfoque digital . . . 31
2.2.2. Teorema de *slice* de Fourier . . . 32
2.2.3. Fotografía por *slice* de Fourier . . . 34
2.2.4. Transformada de Radon de d-planos . . . 38
2.3. Obtención de información tridimensional . . . 40
2.4. Hardware gráfico . . . 42
2.4.1. Paralelismo, arquitectura y paradigma de programación 42
2.4.2. Evolución de las GPUs . . . 46
2.4.3. Programabilidad . . . 49
2.5. Sumario . . . 50

3. FFT sobre GPU **53**
3.1. Trabajos anteriores . . . 53
3.2. *Stream processing* en GPU: *BrookGPU* . . . 55
3.2.1. *Streams* y *kernels* . . . 56

3.2.2. Ejemplo de uso de *kernels* y *streams* en *BrookGPU* . . 58
3.3. Transformada rápida de Fourier 60
3.3.1. De la DFT a la FFT 60
3.3.2. Variantes de la FFT 63
3.3.3. Elección de la variante de Pease 65
3.4. Caso unidimensional, múltiples secuencias de tamaño regular . 68
3.5. Caso bidimensional, método *row-column* 68
3.6. Caso 1D, secuencias grandes alojadas en 2D 70
3.7. Caso propiamente bidimensional 72
3.7.1. Caso bidimensional, $M! = N$ 77
3.7.2. Consideraciones acerca de la variante Pease 2D sobre GPU 80
3.8. Caso bidimensional, descomposición 4-2–2-4 81
3.8.1. Descomposición en bases genéricas 82
3.9. Implementación y rendimiento 87
3.10. Comparativa con otros autores 89
3.11. Aplicación a la Óptica Adaptativa 93
3.11.1. Descripción de un sensor de Shack-Hartmann 94
3.11.2. Recuperación del frente de onda mediante FFT 96
3.11.3. Resultados y análisis 97

4. Transformada apróximada del *FS* 101
4.1. Radon, algoritmos rápidos y desenfoque 101
4.2. Propuesta de modificación de la 2D aDRT de G&D 103
4.3. Análisis 2D y extensión a 3D y 4D 105
4.3.1. Transformada aproximada de Radon en 3D 111
4.3.2. Transformada aproximada de Radon en 4D 112
4.3.3. Complejidad computacional de las aDRTs 112
4.4. Reenfocando digitalmente con una 4D *aDRT* 113
4.5. Transformada aproximada de *focal stack* 115
4.6. Algoritmo de la *aFST* 117
4.6.1. Complejidad computacional 118
4.6.2. Tamaños no potencias de 2 118
4.7. Interpretación óptica de la *aFST* 121
4.7.1. Reordenando la imagen plenóptica 121
4.7.2. Formación de imagen en $s = 0$ 122
4.7.3. Formación de imagen combinando rayos de varias microlentes 126
4.7.4. Cono discreto de desenfoque 131
4.8. Implementación y rendimiento 133
4.9. Obtención de mapas de distancia 135

4.9.1. Enfoque por contraste 137
4.9.2. Enfoque evaluado mediante diferencias 142

5. Conclusiones 149
5.1. Sumario 149
5.2. Publicaciones 151
5.2.1. Contribuciones recogidas en esta Memoria 151
5.2.2. Otras contribuciones, en los mismos tópicos 152
5.2.3. Otras contribuciones, en procesado de señal 154

A. Códigos de ejemplo 155
A.1. Código de la FFT sobre GPU 155
A.2. Código de la aFST 158

B. Derivación del mapeo entre etapas de la *aFST* 163

Índice de tablas

2.1. Modelos de GPUs en el periodo 1999-2008. 47

3.1. Variantes de la FFT. 62
3.2. Código del *kernel* correspondiente a una mariposa *radix-4–radix-2*. 85
3.3. Tiempos de transferencia de datos CPU–GPU sobre PCI-Express. 87
3.4. Tiempos de cómputo de las FFT propuestas. 88
3.5. Comparativa de tiempos de nuestras FFTs 2D con las de otros autores. 89
3.6. Tiempos de ejecución del recuperador de fase 98
3.7. Tiempos de cómputo de la librería FFTW 99

4.1. Mapa de distancias para el máximo contraste en ventanas del volumen focal. 142
4.2. Mapa de distancias para la evaluación de diferencias en ventanas del volumen focal. 146

Índice de figuras

1.1. Mapa de distancia por pixel 2
1.2. Imagen a escala de una GPU y una CPU 8

2.1. Cámara oscura. 14
2.2. Esquema de una lente convergente. 15
2.3. Relación entre apertura y profundidad de enfoque. 16
2.4. Pirámides de visión. 17
2.5. Configuraciones plenópticas básicas 18
2.6. Radiancia de un objeto rodeado por un recinto 3D 19
2.7. *Lightfield* parametrizado mediante 2 planos. 20
2.8. Uso de la profundidad para interpolar rayos. 21
2.9. Cámara de superficie, *scam*. 22
2.10. Relación entre rayos en (x, z) y puntos en (s, u) 23
2.11. Soporte espectral de una escena lambertiana. 24
2.12. Curvas de tasa óptima de muestreo. 25
2.13. Array de cámaras de Stanford 26
2.14. Análisis plenóptico de una cámara estenopeica. 27
2.15. Principio de funcionamiento de una microlente. 28
2.16. *Lightfield* 4D modulado sobre un sensor 2D. 30
2.17. Reparametrización a un plano virtual. 31
2.18. Teorema generalizado del *slice* de Fourier. 33
2.19. El transporte de luz equivale a una cizalla, o *shear*, de la radiancia. 34
2.20. Fotografía por *slice* de Fourier. 36
2.21. Geometría de la Estereovisión. 40
2.22. Evolución prevista de los procesadores. 44
2.23. En verde, espacio dedicado a cómputo en una CPU y en una GPU. 45
2.24. Evolución del rendimiento de CPUs y GPUs. 48

3.1. Mapeado directo entre *strels* de entrada y salida. 56

3.2. Mariposas de las variantes Cooley&Tukey, y de Pease. 65
3.3. FFT de múltiples secuencias 1D. 66
3.4. FFT 1D de una secuencia alojada en un *stream* 2D. 71
3.5. *Strels* involucrados en una mariposa Pease radix-2–radix-2 . . 74
3.6. FFT 2D, Pease *radix-2–radix-2*, en términos de *streams* y *kernels*. 75
3.7. Secuencia 2D 4×4 en un *stream* de `float4`. 77
3.8. Operaciones de mapeo para computar la FFT 2D de una secuencia 4×4. 78
3.9. Distancias lsb_x, MSB_x, lsq_y y MSQ_y en una FFT *radix-4–radix-2*. 86
3.10. Operaciones de *stream* que realizan una mariposa Pease *radix-4–radix-2*. 86

4.1. Combinación multiescala de segmentos de línea. 106
4.2. Sumas involucradas en el cómputo de un valor en la 2D aDRT. 107
4.3. Sumas en una 2D aDRT sobre una rejilla 4×4. Planteamiento. 108
4.4. Sumas en una 2D aDRT sobre una rejilla 4×4. Desarrollo. . . 109
4.5. Combinaciones entre nodos vecinos en la etapa inicial de la aDRT en 2, 3 y 4 dimensiones. 110
4.6. Reenfoque como suma a lo largo de hiperplanos de un *lightfield*. 114
4.7. Definición recursiva de líneas discretas con bases 2 y 3. 119
4.8. Líneas discretas para N=8 y base binaria; y N=9 con base ternaria. 120
4.9. Distribución de rayos en el interior de una plenóptica. 121
4.10. Orden de BRDFs de una plenóptica. 123
4.11. Orden de subaperturas de una plenóptica. 124
4.12. DOF comparados de una imagen de subapertura y de la imagen central del *focal stack*. 125
4.13. Interpretación óptica de la *aFST* para *s = 1*. 127
4.14. Zonas exclusivas en la 2D aDRT. 128
4.15. Interpretación óptica de la *aFST* para *s = 2*. 130
4.16. Pirámide de desenfoque discreto. 132
4.17. *Focal stack* computado con la aFST. 134
4.18. Mapa de distancias para el máximo valor del volumen focal. . 136
4.19. Medida del enfoque en una ventana de rayos. 138
4.20. Mejores imágenes de máxima profundidad de campo para los métodos propuestos. 147

Resumen

Las cámaras plenópticas preservan las componentes angulares y posicionales de la información visual de una escena, aumentando las posibilidades de procesamiento tras la exposición respecto a lo ofrecido por una cámara convencional.

En esta tesis se estudia el procesamiento para el enfoque digital, a posteriori, de una imagen plenóptica. Este enfoque se puede realizar sobre la misma ubicación en que sucedería de tratarse de una cámara convencional, o bien en planos desplazados hacia delante o hacia atrás sobre el eje óptico, generando lo que se conoce como un volumen focal de la escena.

Cuando se pretende analizar esta totalidad de planos de reenfoque, en lugar de solo uno, es conveniente utilizar algoritmos rápidos. Con tal fin surgió la técnica de fotografía por *«slice de Fourier»*, que conlleva la transformación de la imagen plenóptica 4D al dominio de Fourier, y la posterior conformación de planos de imagen 2D reenfocados mediante un procesamiento en el dominio transformado y posterior inversión.

Esta tesis contribuye a la aceleración de dicho procesado proponiendo el cómputo del algoritmo básico de la transformada rápida de Fourier sobre hardware masivamente paralelo y de bajo costo, las unidades de procesamiento gráfico, GPU. Se concluye que la aplicación de la variante de Pease y una descomposición adecuada de las bases de cómputo permite la aceleración de la FFT sobre esta plataforma. Esta reducción en los tiempos de cómputo se aplica en el campo de la óptica adaptativa, lográndose la recuperación modal de la fase del frente de onda de un sensor Hartmann-Shack en una fracción del tiempo requerido de realizarse sobre una CPU.

Como aún con la aceleración lograda con la FFT no se puede garantizar el procesamiento de un volumen focal en tiempos de adquisición de video, se propone un método computacionalmente menos costoso para hallar con una sola transformación el volumen focal. Para ello se ha extendido la transformada discreta de Radon, hasta ahora solo planteada en 2 y 3 dimensiones, para que partiendo de una imagen plenóptica 4D obtenga directamente como resultado un volumen focal 3D. El algoritmo que se obtiene reduce más de

un 90 % los tiempos del mejor algoritmo disponible hasta el momento.

Adicionalmente se realizan consideraciones sobre la posibilidad de extraer un mapa denso de distancias al analizar las profundidades en que los distintos objetos presentes en la escena se enfocan dentro del volumen focal.

Abstract

A plenoptic camera preserves angular and positional information of the rays composing the light field of a scene. This preservation increases the chances of image processing after the exposure, with respect to a conventional camera.

This thesis studies the processing of plenoptic images for digital refocusing. This refocusing can be performed on the same location in which it would happen on a conventional camera, or in planes displaced forward or backward along the optical axis, generating what is known as a focal stack of the scene.

When one tries to analyze the focal stack as a whole, instead of only one plane, it is convenient to use fast algorithms. For this purpose the *slice Fourier photography* technique was developed by other authors. It involves the transformation of the 4D plenoptic image to Fourier domain, and subsequent formation of 2D images refocused by signal processing on the transformed domain and inverse transform.

This thesis contributes to the speedup of the aforementioned processing by adapting the fast Fourier transform algorithm in order to be computed on a low-cost massively parallel hardware: the graphics processing units, GPUs. It is concluded that the application of the Pease variant and a proper decomposition of the bases of computation enables the speedup of Fourier transform on this platform. The acomplished reduction in computation times is applied in the field of adaptive optics, achieving the modal recovery of wavefront phase of a Hartmann-Shack sensor in a fraction of the time required to perform it on a CPU.

Even using our GPU implementation, Fourier methods for plenoptic processing can not guarantee the computation of a focal stack within video acquisition rates. Therefore we present a less demanding approach to find a focal stack in a single transformation step. The proposed transform extends discrete Radon transform, up to now only available for 2 and 3 dimensions, in order to cope directly with a 4D plenoptic image and to generate its associated 3D focal stack, in a single step. Our proposal reduces up to 90 % the

times achieved by the best algorithm available so far.

Finally, considerations are given on the possibility of extracting a dense depth map by analyzing the position in which the focusing of the various objects take place within the focal stack.

Capítulo 1

Introducción

En el presente capítulo se expondrán los motivos que fundamentaron, y las circunstancias que rodearon, la realización de esta investigación, se introducirá una visión general del problema a tratar, y se iniciará la discusión respecto a qué puntos han centrado nuestra atención y cómo hicimos para avanzar en la solución del problema. Por último se expondrá cómo se han distribuido los contenidos de esta memoria.

En el título de esta memoria de Tesis Doctoral se pueden distinguir cuatro elementos claves. El primero hace referencia a la extracción de información tridimensional respecto de la conformación de una escena. Éste fue el problema sobre el que hemos tratado de avanzar con nuestra investigación. Las restantes tres partes del título constituyen las herramientas sobre las se basa la solución propuesta: los sensores plenópticos, mediante los cuales se capta pasivamente la información visual de la escena; el procesado de señales, mediante el cual se extrae de la información plenóptica conclusiones respecto a la forma de la escena observada; y el hardware gráfico, mediante el cual se acelera los cómputos necesarios.

Para entender qué ha motivado la investigación realizada es necesario entender qué significa obtener información tridimensional de una escena. Para ello se presenta la figura 1.1, en la que se muestra una escena y, en el recuadro interior, su mapa de distancias, en el que se ha estimado la distancia que separa el punto de observación de las superficies de los objetos que conforman la escena.

Obtener un mapa de distancias como el mostrado supone resolver, casi completamente, el problema de la visión por ordenador. David Marr, en su libro *«Vision: A Computational Investigation into the Human Representation and Processing of Visual Information»*, denominó tal representación *«2.5D Sketch»*, la imagen mental ciclópea que el cerebro humano crea a partir de las dos imágenes retinianas y a las cuales dota de profundidad principalmente por

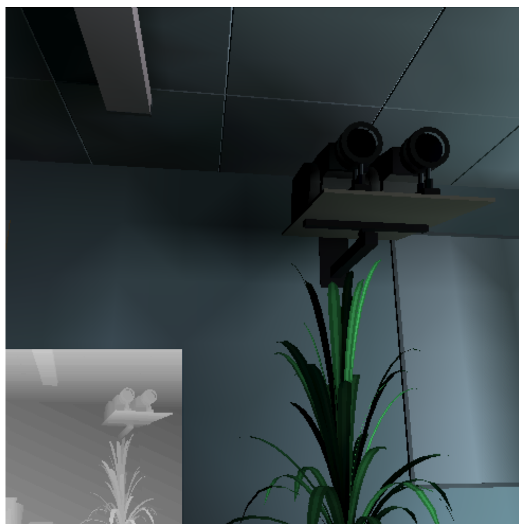


Figura 1.1: Mapa de distancia por pixel, o *«sketch 2.5D»*, en un simulador de escenas virtuales. En el recuadro interior se representan en intensidad las distancias de los distintos puntos de la escena al observador.

«stereopsis», y que se considera el paso previo a la imagen mental plenamente 3D, la que nos permite conocer qué objetos hay en el mundo y dónde se sitúan. Nuestra principal motivación ha sido dar solución a dicho problema, ya que lograrlo, siquiera parcialmente, sería de gran utilidad en muchos campos.

De entre ellos nos hemos centrado en su aplicación al campo de la *«3DTV»*. Un campo aún emergente, que concibe un nuevo tipo de televisión. Una donde las imágenes dejan de ser planas, igual que en su momento dejaron de ser en blanco y negro, y pasan a estar dotadas de la sensación de tridimensionalidad. Para ello es necesario recrear, emitiendo desde una superficie plana, una información visual que sea, a ojos del espectador, indistinguible de la que se obtuviese al observar directamente la escena en cuestión. Quizás recrear esta ilusión es el menor de los problemas implicados: también es necesario captar, codificar adecuadamente y transmitir esta información que aporta la componente tridimensional que descartan las cámaras y pantallas convencionales. Los avances y el grado de madurez tecnológica alcanzados en los campos de la tecnología de representación 3D autoestereoscópica, la codificación y trasmisión de vídeo digital y los algoritmos de visión por computador implicados, han posibilitado el desembarco de esta tecnología en el ámbito del hogar.

Curiosamente, para esta aplicación —que destacaremos del resto, y que fijará los requisitos de calidad y velocidad con que pretendemos conseguir dichos mapas de distancia—, no es estríctamente imprescindible disponer

explícitamente de la información geométrica de la escena. Se podría lograr la ilusión de tridimensionalidad representando una escena a partir de imágenes de la misma captadas desde diversos puntos de vista, esto es, una representación pura de imágenes, sin información geométrica. De esta manera se conocería en origen cómo varía la escena al cambiar el punto de vista y se podrían recrear esas ligeras variaciones en destino, haciendo llegar imágenes diferenciadas y adecuadas a cada uno de los ojos del espectador de tal manera que se le indujese a pensar que lo que tiene ante sí ya no es una imagen bidimensional plana al uso. Además, y dependiendo del rango de puntos de vista captados, codificados, transmitidos y recreados se podría permitir cierta libertad de movimiento del espectador y que la sensación ilusoria persistiese, o incluso la presencia simultánea de más de un observador. Éste método de representación basado en imágenes es muy exigente en sus requerimientos, y lo es tanto más cuanto más crece el número de puntos de vista empleados. Por ejemplo la transmisión de 16 puntos de vista como señales de vídeo sin comprimir de 1280x720 píxeles, con submuestreo cromático 4:2:0 y 30 fps requeriría unos 4.95 gigabits por cada segundo, algo prohibitivo para las tecnologías actuales. Por lo tanto, aunque no es necesario sí es conveniente, disponer de información geométrica adicional, que permita recrear en destino esa misma ilusión cuando se dispone de una señal de vídeo convencional a la que se acompaña la información contenida en el mapa de distancias. Este método de representación aumenta los requisitos de cómputo en origen y destino, pero se disminuye, para el ejemplo anterior, a unos 530 megabits la tasa de información por segundo en el canal de transmisión, que aún se puede rebajar si se consideran menos de 8 bits para codificar las distancias.

Nos hemos marcado como objetivo llegar a implementar en un prototipo un sistema que pueda alimentar a una pantalla 3D autoestereoscópica con dos restricciones adicionales muy importantes, a saber, que el sistema ha de estar compuesto por una sóla cámara, y que el procesado ha de ser en tiempo real. El problema, sin estos requisitos adicionales, ya ha sido resuelto. Entiéndase resuelto en la medida en que aún no siendo unívocamente invertible el proceso de determinación de las formas tridimensionales a partir de su proyección a imágenes planas, sí es posible determinar una, la más apropiada, de las configuraciones tridimensionales entre aquéllas que pueden dar lugar a la información captada.

1.1. Visión de conjunto

En esta sección se describe una visión de conjunto del entorno del problema para así estar en disposición de formular la hipótesis en que sustentaremos

nuestra propuesta de solución.

Como se ha avanzado, estará fundamentada en el uso de tres tecnologías: los sensores plenópticos, el procesado de señal y el hardware gráfico como recurso computacional.

1.1.1. El sensor

Se denomina *percepción* a la capacidad para conocer nuestro entorno y así poder interaccionar con él. Por ejemplo, para desplazarnos hemos de conocer las estructuras que componen el entorno y los eventos que en él se producen. Accedemos a dicha información captando uno o varios de los tipos de energía presentes a nuestro alrededor. Con diferencia, la que más información nos aporta es la luz, la radiación electromagnética en el espectro visible, la cual captamos con nuestros ojos y que por el complejo proceso de interacción ojo-cerebro que supone la *visión*, termina por facilitarnos la información que requerimos para realizar esa y otras tareas. Replicar este proceso en una máquina es tremendamente laborioso.

En la primera etapa de este proceso se ha de decidir cómo captar la información, puesto que no viene predeterminado como sucede con el sistema binocular humano. ¿Es lo más adecuado replicar la configuración en que se basa la visión humana? Parece factible, puesto que nosotros «naturalmente» podemos formar mentalmente ese esbozo 2.5D, trasladar con éxito esa configuración binocular a una máquina. Sin embargo, la visión por ordenador, pese a haber logrado avances notables, no termina de dar una respuesta definitiva siguiendo esta vía. Por este motivo, y unido a él, el objetivo asumido de que el sistema cuente con una única cámara, conviene revisar la idea de «energía radiante del entorno», antes de plantear cómo captarla de la forma más adecuada para nuestros fines.

En palabras de Adelson y Bergen, en su artículo de 1992,

> «The world is made of three-dimensional objects, but these objects do not communicate their properties directly to an observer. Rather, the objects fill the space around them with the pattern of light rays that constitutes the plenoptic function, and the observer takes samples from this function. The plenoptic function serves as the sole communication link between the physical objects and their corresponding retinal images. It is the intermediary between the world and the eye.»

En otras palabras, la energía electromagnética es emitida, reflejada y refractada por las superficies de los cuerpos que componen el entorno, rellenando el espacio con rayos de luz. Considerados en su conjunto constituyen

una señal de siete dimensiones. A saber, la posición espacial 3D donde se ubica el punto de observación, la dirección 2D angular de observación, la longitud de onda y el instante de tiempo. Ésta es la información disponible midiendo el flujo de energía radiante, la denominada *función plenóptica*: $P(V_x, V_y, V_z, \theta, \phi, \lambda, t)$. Olvidándonos un momento de la longitud de onda, vemos que nuestro sistema binocular en cada instante de tiempo está accediendo a dos posiciones determinadas de entre todas las posibles del espacio de observación, y desde ellas solo se capta información en un rango limitado de las dos dimensiones angulares que aún restan. Por lo tanto, de toda la información disponible inicialmente, nos quedamos con dos muestras, y aún limitadas, de la función plenóptica. Dos imágenes bidimensionales al uso: $P(\theta, \phi)|_{\mathbf{V}=\mathbf{V_0}=\{V_{x0}, V_{y0}, V_{z0}\}, t=t_0}$ y $P(\theta, \phi)|_{\mathbf{V}=\mathbf{V_1}, t=t_0}$. [1]

Las variables que habíamos sacado de la función plenóptica se pueden reincorporar a ella. La componente temporal fácilmente: en ese caso hablaríamos de secuencias de vídeo. En cambio, incorporar más posiciones de observación no es tan sencillo. De esto es de lo que se encargan los denominados *sensores plenópticos*. A diferencia de nuestros ojos, y su trasunto artificial, las cámaras convencionales, estos sensores captan al menos 4 dimensiones de la función plenóptica, obteniendo una variedad de ángulos de observación, no ya para un punto prefijado, sino para una cantidad de ellos, normalmente ubicados sobre un plano: $l(V_x, V_y, \theta, \phi)|_{V_z=V_{z0}, t=t_0}$.

1.1.2. El procesado

El concebir un sensor tal, ¿contribuye a avanzar en el problema central de la extracción de la información de la escena? Vayamos por pasos. Como veremos al describir los sensores plenópticos, se seguirá captando la luz sobre un sensor 2D, por lo que hemos de llegar a un compromiso al muestrear simultáneamente las componentes angulares y espaciales de la función plenóptica 4D. Para conocer más acerca de las unas, se sacrifica la capacidad de conocer en detalle las otras. Por lo que cambiar el sensor no resulta exactamente en un aumento de la información disponible, sino más bien en un cambio en la naturaleza de la misma. Luego, es de esperar que, sustituido el sistema de captación binocular, el procesado a considerar ya no sea la estereovisión.

Antes de apuntar cómo procesar esta señal plenóptica 4D, volvamos a un punto sobre el que se pasó muy rápidamente: la afirmación tácita de que los ojos captan dos muestras puntuales de la señal plenóptica. Esto no es así.

[1] Nótese que lo que estos autores tratan como resolución angular de la incidencia de radiación en la función plenóptica equivale a la resolución espacial en las cámaras e imágenes convencionales en las que se define una focal.

Sería cierto, si nuestra visión fuera idealmente *estenopeica*.[2] Esto es, que el ojo dejara pasar para cada dirección sensada exactamente un rayo de luz. En cambio, para aumentar la cantidad de luz incidente, la apertura de la pupila no es puntual.

Una cámara oscura y una apertura no puntual no dan lugar a la formación de imágenes nítidas. Por lo que es necesario contar con un elemento adicional, encargado del proceso de *enfoque*. Este papel en el ojo lo juegan la córnea y el cristalino, y en las cámaras fotográficas la lente objetivo. Esta parte óptica se encarga de hacer converger sobre un punto de la superficie fotorreceptora la luz que provino exclusivamente de un punto del espacio, restableciendo así la nitidez de la escena, pero no de toda, sino de aquellas zonas situadas a una distancia tal sobre el eje óptico que la curvatura que presenta el frente de onda de la luz proviniente de estas zonas al alcanzar la apertura es la inversa a la curvatura que adopta en ese momento el sistema óptico. La luz proviniente desde esa distancia converge exactamente sobre el sensor, dando lugar a una imagen enfocada, mientras que la luz proviniente de distancias mayores o menores a la de enfoque se dispersa sobre el sensor dando lugar a regiones desenfocadas en la imagen.

Nótese que este proceso no requiere el concurso de ambos ojos, y que existe una componente mecánica, fisiológica. Son la contracción o dilatación del músculo ciliar las que varían la potencia refractiva del cristalino y, con ello, la distancia a la cual se enfoca. A este proceso se le denomina *acomodación*. Y da lugar a una de las pistas con las que el cerebro cuenta a la hora de inferir la distancia a los objetos de la escena.[3]

Al problema de extracción de distancias le habíamos añadido dos condicionantes. Que la solución emplease una sola cámara y que el procesado fuera en tiempo real. Llegados a este punto estamos más cerca de poder enunciar la hipótesis sobre la que trataremos de avanzar. Hemos visto que dependiendo de la curvatura que adopta en cada instante el sistema optomecánico formado por el cuerpo ciliar, la cornea y el cristalino se perciben como enfocados únicamente aquellos objetos situados a una distancia determinada, pero que, de cambiar esa curvatura, los objetos enfocados serían otros. Por lo tanto, a

[2]El pequeño orificio de las cámaras que carecen de lentes es el *estenopo*. Es habitual el uso del término inglés: *«pinhole»*.

[3]Cuando el cristalino se encuentra en dilatación se *enfoca al infinito*, y la imagen producida es nítida para aquellos objetos situados a más de 6 metros de distancia. La acomodación juega su papel a la hora de determinar distancias más cercanas. Otra pista preponderantemente fisiológica, y muy unida a la acomodación, es la *convergencia*, la capacidad para entornar los ojos y hacerlos coincidir sobre objetos cercanos. La convergencia implica hacer uso de ambos ojos, lo mismo que en la *«stereopsis»*, que supone la pista fundamental, y en la que, en cambio, no existe componente fisiológico.

la pupila está llegando la información necesaria para barrer todo un espacio de enfoque, y esa información no es otra que la función plenóptica 4D registrada sobre la superficie de la pupila. Luego, si consideramos que es posible cuantificar la cantidad de enfoque que tiene una región de una imagen, un sensor plenóptico puede sustituir a una configuración binocular a la hora de estimar a qué distancia está un cierto objeto. El problema es que seguimos teniendo un sistema optomecánico realizando la tarea de enfoque, y éste es selectivo, capaz de ofrecer solamente una imagen enfocada a una distancia determinada en cada instante de tiempo.[4]

Por todo ello, reformularemos el problema inicial de extracción de distancias a partir de una sola cámara/ojo como el problema de procesar digitalmente la función plenóptica 4D de tal manera que se replique por medios computacionales el proceso de enfoque que venía jugando un sistema optomecánico, y se obtenga la solución no para una distancia determinada, sino para tantas como el compromiso espacio-angular adoptado en el sensor plenóptico permita. Ese cómputo, de ser realizable, además ha de serlo conforme a los requisitos de tiempo que fijamos en nuestro segundo compromiso: el procesado en tiempo real para alimentar a un monitor 3D, lo que pone una cota temporal máxima de tiempo de cómputo de 40 ms.

Nótese que el proceso de reenfoque es un proceso que no se ve afectado por la disposición de los objetos en la escena. Para enfocar a una cierta distancia basta con «torcer» adecuadamente los rayos. Independientemente de si hay o no objetos a esa distancia o si existen oclusiones, la imagen obtenida es «correcta»: es la que se hubiera obtenido de haber dispuesto una lente y sensor convencional en lugar de un procesado digital sobre un sensor plenóptico. Haciendo un símil con los procesos involucrados en el estéreo computacional, podemos decir que el reenfoque es determinista y unívoco, como lo son la rectificación epipolar o la reconstrucción geométrica para un emparejamiento dado en el estéreo computacional; a diferencia de ellos, la elección de correspondencias entre píxeles del par de imágenes estéreo y la estimación de la superficie de mejor enfoque sobre un rayo del proceso de visión monocular por enfoque, son ambiguos y no siempre resolubles.

Entraremos en profundidad en este procesamiento en la segunda sección del próximo capítulo. Como veremos, este problema ha sido un tema relevante para la comunidad científica mientras se llevó a cabo la presente investigación y sobre él se han realizado avances importantes por parte de diversos grupos de investigación. Se podría afirmar otro tanto, solo que quizás en menor

[4]Existe toda una rama dedicada a estimar las distancias de los objetos de una escena a partir de dos o más imágenes correspondientes al mismo instante de tiempo aunque enfocadas a distintas distancias. Para captar tales medidas se suelen usar divisores de haz y múltiples sistemas ópticos independientes.

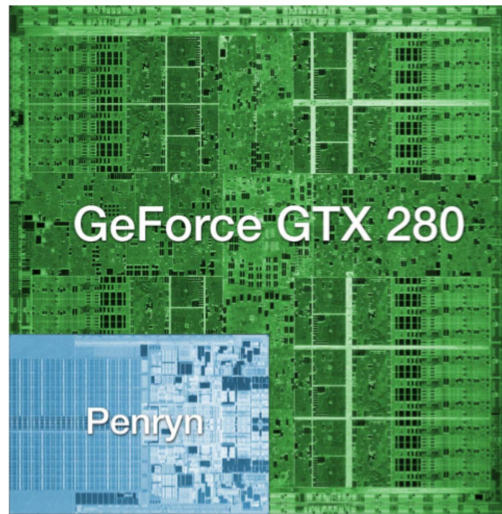


Figura 1.2: Imagen a escala de una GPU y una CPU, ambas de última generación. La GPU está fabricada en una tecnología de 65 nm, mientras la CPU lo está en 45 nm. La GPU en sus 1.4 billones de transistores implementa 240 núcleos de cómputo y 316 KB de cache L1+L2. La CPU, un *«Dual Core Itanium 2»*, cuenta con 24 MB de memoria caché L3, a los que destina 1.5 de sus 1.7 billones de transistores.

medida, de los sensores plenópticos y del hardware gráfico.

1.1.3. El recurso computacional

Comencemos por estimar los requerimientos de cómputo del problema enunciado anteriormente. Tomemos el sensor de luz de mayor resolución que incorpora actualmente una cámara portátil[5] de vídeo: 12 MPx, unas 4500 líneas verticales.

Teniendo en cuenta que cada pixel del sensor codificará un rayo de la función plenóptica, que se recibirán cada segundo 25 ó 30 imágenes, y que se considerarán los 3 canales de color, la cantidad de datos a procesar será del orden de 10^9 muestras por segundo. Lo que da idea de lo conveniente que resultaría que existiese un algoritmo de complejidad lineal o cuasi-lineal. Puesto que además debemos contemplar que parte del tiempo entre imágenes será consumido en las lecturas de datos, transferencias de memoria y el envío al monitor 3D. Parece que el recurso computacional a emplear, al igual que el sensor y el procesamiento elegidos, no podrá ser uno convencional.

[5]El término inglés *«hand-held»* es mas específico que únicamente portátil.

Las tarjetas gráficas de nueva generación, denominadas *«Graphical Processing Units»*, GPUs, trasladan al sector de consumo la calidad visual de la síntesis de imagen tridimensional fotorrealista, esforzándose en permanecer en tiempos de ejecución interactiva. La potencia de cómputo demandada exige procesar del orden de cientos de millones de vértices y píxeles por segundo. Las nuevas generaciones de tarjetas gráficas no sólo han elevado en varios órdenes de magnitud la potencia computacional de las generaciones previas, superando con ello en prestaciones a las CPUs, sino que además presentan un modelo de programación cada vez más abierto. El salto cuantitativo, en cuanto a potencia de cómputo, y el salto cualitativo, en cuanto a libertad de programación de los recursos del hardware gráfico, han promovido un marcado crecimiento en el sector del ocio informático, lo que a su vez realimenta a la industria de hardware para gráficos, empujando cada vez más allá a esta tecnología. Así, el motor gráfico que incorpora la tarjeta *«NVIDIA GeForce GTX 280»*, lanzada al mercado en el verano de 2008, alcanza los 933 Gflop/s[6]. Este potencial, en un periférico de bajo coste y de uso extendido, ha llamado la atención de la comunidad científica, que se las ha ingeniado para realizar cómputos genéricos sobre una plataforma diseñada en principio para tareas específicas. En la figura 1.2 se aprecian a escala los dos productos más punteros de una y otra clase.

Su desarrollo como hoy se las conoce se remonta a finales de 2001, aunque con un punto de inflexión en torno a 2005 y aún otro en 2007. En tan corto espacio de tiempo han sido aplicadas con éxito en multitud de campos. Entre ellos el cómputo de transformadas discretas, el álgebra lineal, las ecuaciones diferenciales, las técnicas multi-grid, la dinámica de fluidos, el crecimiento cristalino, la planificación de movimiento, el trazado de rayos, el cálculo de radiosidad, y un largo etcétera.

La diferencia en rendimiento respecto al de arquitecturas convencionales se logra haciendo predominar las unidades de cómputo frente a *«caches»* y lógica de control sobre la superficie del chip. Relajando la complejidad de la lógica de programa es posible aprovechar en mayor medida el paralelismo implícito en cierto tipo de problemas los cuales se pueden beneficiar del uso de cientos de *«cores»*. Es el caso de la síntesis de imagen tridimensional a partir de una descripción geométrica. Cada primitiva geométrica —y pueden ser millones las que compongan una escena— debe ser proyectada en pantalla realizando una serie de transformaciones de punto de vista lo que implica realizar, para cada vértice e independientemente del resto, las mismas multiplicaciones de vector-matriz, y sin importar cuál es el orden en que se las procese.

[6]Frente a los 48 Gflop/s de un *«Core 2 Quad»* a 3 GHz

El proceso de adaptarse específicamente a este tipo de problemas desembocó en el desarrollo de un tipo de arquitectura, a la cual se denomina *«single-instruction, multiple-thread»*, *SIMT*, que según ha ido evolucionando y expandiéndose para hibridarse con la mayor generalidad de las arquitecturas convencionales, ha dado lugar a un recurso computacional muy a tener en cuenta por los desarrolladores en el ámbito del cómputo científico. Éste será el recurso computacional que se escogió estudiar para realizar sobre él el procesado digital de la función plenóptica.

1.2. Hipótesis

Se propone el estudio del procesamiento de señales multidimensionales focalizado en el desarrollo e implementación de algoritmos que permitan la obtención de información tridimensional en escenas, utilizando sensores plenópticos y empleando como recurso computacional la tecnología de las GPUs.

Específicamente se pretende obtener a partir de una función plenóptica 4D espacio-angular un volumen espacial 3D de imágenes enfocadas, donde se haga explícito el parámetro de distancia, z, a lo largo del eje óptico. Se pretende que este algoritmo sea de complejidad lineal o cuasi-lineal con el tamaño de los datos, y que su óptima implementación sobre GPU permita el procesado en tiempo real.

1.3. Objetivos

Para avanzar en la hipótesis recién enunciada, se plantean los siguientes objetivos parciales:

1. El dominio del sensor elegido, el estudio de su viabilidad para los objetivos de la Tesis y su caracterización desde el punto de vista de la teoría de la señal;

2. el estudio de la función plenóptica y las variaciones que en ésta producen los cambios de profundidad de los objetos en la escena;

3. y, el dominio de la tecnología de la GPU y el procesado de señales —con un especial énfasis en el cómputo de la transformada rápida de Fourier, *FFT*, u otras análogas, con el fin de lograr complejidad lineal o cuasi lineal— sobre esta plataforma tecnológica;

1.3.1. Ventajas del enfoque adoptado

Nuestra hipótesis presenta una serie de ventajas respecto a otras soluciones:

1. Capacidad de hacer medidas de enfoque pasivo, monocular, determinista y libre de elementos mecánicos, lo que hace abordable el objetivo del procesado en tiempo real;

2. aprovechamiento novedoso y práctico de la creciente resolución espacial de los chips sensores de luz;

3. y, el aprovechamiento de la creciente disponibilidad y grado de perfeccionamiento de los monitores 3D, mientras se sigue haciendo uso de la mayor parte de la infraestructura existente en cámaras, sus ópticas, y las técnicas de codificación y transmisión de vídeo.

1.3.2. Aplicaciones y objetivos secundarios

Lo que estamos proponiendo es, en otras palabras, la realización de tomografía del espacio objeto de un sensor. De confirmarse la hipótesis, su aplicación en instrumentación astronómica sobre un sensor de frente de onda —muy similar a los sensores plenópticos— permitiría barrer la totalidad de la atmósfera allí donde las técnicas de *óptica multiconjugada* únicamente permiten la elección *a priori* de unas pocas capas. De demostrarse viable esta aplicación, podría ser clave en la caracterización y corrección de la perturbación atmosférica.

Pero no sólo permitiría nuevas aplicaciones, sino también reforzar aplicaciones bien conocidas aplicando los resultados del tercer objetivo: la obtención de un algoritmo de FFT especialmente adaptado a las peculiaridades de las GPUs. Así, el uso de las técnicas de Óptica Adaptativa, que se ve limitado por el tiempo de estabilidad atmosférico, de 10 ms en el rango visible, hace necesario recurrir a hardware muy especializado para realizar los cómputos necesarios en ese margen temporal. El dominio de la tecnología de la GPU permitiría la recuperación del frente de onda para sensores complejos dentro de estos márgenes temporales.

Asimismo, el objetivo de lograr *metrología robusta* en interiores no está ni mucho menos ceñida a su aplicación a 3DTV, sino que sus usos y beneficios en aplicaciones industriales y biomédicas serían numerosísimos. Por ejemplo jugaría un papel clave en los sistemas de sustitución sensorial para invidentes.

1.4. Organización de esta memoria

Las mejoras sugeridas al trabajo de Ren Ng se recogen de manera diferenciada en los capítulos III y IV. En el capítulo III se profundiza en la implementación de algoritmos rápidos de la transformada de Fourier, y en el IV en una propuesta de reenfoque sin pasar por el dominio frecuencial. Con anterioridad se presentan en el *Marco Teórico* los conceptos necesarios para entender los aportes propios. En ese capítulo se sitúa también una contribución a medias entre el doctorando y otros miembros del grupo de investigación. Con esto se quiere separar lo que es estrictamente autoría del doctorando, guiado por sus directores de tesis, de lo que es coautoría. Otros aspectos investigados en el transcurso de la realización de esta Tesis son dejados en un segundo plano, con el objeto de dotar de coherencia el desarrollo de la misma, aunque se repasan sucintamente en el apartado *Publicaciones* del último capítulo. Igualmente se desatiende el orden temporal de las investigaciones a la hora de presentar un hilo discursivo coherente.

Los resultados de aplicar las contribuciones desarrolladas a los campos de la Óptica Adaptativa y a la obtención de mapas de distancia se encuentran incluidos en los capítulos correspondientes a cada una de las dos contribuciones esenciales de esta Tesis. La presente memoria se cierra con un último capítulo de *Conclusiones*.

En los apéndices se ofrece el pseudo-código que permite conocer los detalles de implementación de los algoritmos propuestos.

Capítulo 2

Marco teórico

Este capítulo hace un repaso de los conceptos en que se sustentará la solución propuesta al problema de extracción de distancias de una escena. Se comienza incidiendo en el campo de la fotografía computacional y, en particular, en el concepto de función plenóptica y en los sensores plenópticos que permiten medirla. Se describen los diversos modos de realización de cámaras de un solo cuerpo que permiten sensar la información angular que desechan las cámaras convencionales. A continuación se describe el método de fotografía por *«slice» de Fourier* y sus implicaciones. Se propone la transformada de *«focal stack»*[1], como la transformada integral discreta adecuada para el cómputo de tal volumen a partir de la función plenóptica 4D. Se hace entonces un repaso breve a las técnicas que permiten evaluar cuán enfocada está cada región de una escena a partir de su *«focal stack»*. Y se concluye el capítulo repasando la evolución y estado actual de las GPUs.

2.1. Fotografía computacional

La fotografía computacional es el campo donde converge el uso de la óptica con el del ordenador para aumentar las capacidades de la fotografía convencional. La óptica empleada en este tipo de fotografía permite capturar los rayos de maneras inusuales —por ejemplo, manteniendo la información angular de la radiancia— y el ordenador permite procesarlos de maneras igualmente inusuales.[2] Con ello se logra «procesar ópticamente» antes de registrar los rayos y procesar computacionalmente antes de «revelar» la imagen resul-

[1] Denominaremos *«focal stack»* al volumen constituido por las imágenes enfocadas a distintas distancias sobre el eje óptico

[2] No entraremos en las posibilidades que surgen de modificar la iluminación de la escena, que es otro amplio tema de estudio en fotografía computacional.

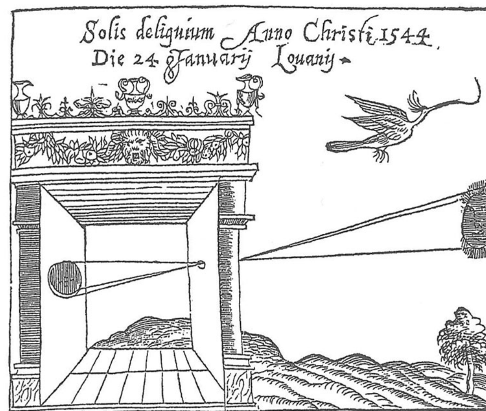


Figura 2.1: Cámara oscura.

tante. Para así ensanchar las posibilidades de experimentar nuestro mundo visual. Ejemplos de esto son las imágenes de apertura sintética, las imágenes de alto rango dinámico, o las imágenes multiespectrales. Finalmente, se puede capturar sobre los mismos sensores usados en fotografía convencional, información relevante en torno a la geometría de la escena fotografiada, que es desechada con los métodos convencionales, y que es de importancia para nuestro estudio. Por ello abordaremos algunas de las definiciones básicas de la fotografía convencional.

Denominamos luz a una parte del espectro de la radiación electromagnética cuyas longitudes de onda están comprendidas entre los 380 nm y los 770 nm y que da lugar al concepto de visión.

Ligada a la visión aparece la fotografía. «Escribir con luz», si atendemos a su etimología griega: captar y fijar la sensación que produce la luz sobre una superficie fotosensible.

Ya Aristóteles en el siglo V a.C. hacía referencia al fenómeno de la *«camera obscura»*, aunque ese término fue acuñado por Johaness Kepler en su obra *Ad Vitellionem paralipomena* de 1604. Al practicar un pequeño orificio, o estenopo, en un recinto cerrado se obtiene, en la pared interior enfrentada al orificio, una imagen invertida de los objetos situados en el exterior, como se ilustra en la figura 2.1. Este fenómeno se explica por la propagación rectilínea de la luz. La imagen formada está constituida por todos aquellos *rayos* que van de un punto de la escena a un punto del interior de la cámara cruzando a traves del orificio. La idea intuitiva de rayo es común para explicar la propagación de la luz. Más formalmente, la óptica geométrica define un rayo como el haz de luz idealmente estrecho que se propaga en línea recta en un medio homogéneo y cuya dirección es perpendicular al avance del frente de onda.

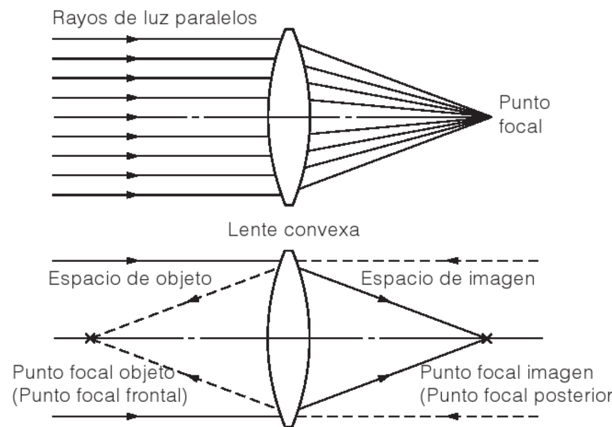


Figura 2.2: Esquema de una lente convergente.

Aún cuando en esta simplificación no se considera la naturaleza ondulatoria de la luz, su uso permite explicar de manera sencilla su propagación en ausencia de los fenómenos de difracción e interferencia. Esto es, considerando exclusivamente las leyes de reflexión y refracción.

Las dos características esenciales en que difiere una cámara fotográfica moderna de tales dispositivos son: la capacidad de fijar la imagen por medio de película química o dispositivos fotoeléctricos; y el uso de una lente objetivo que incrementa el tamaño de la apertura, y con ello la luminosidad mientras se mantiene la nitidez de las imágenes.

Lentes

Una lente es un dispositivo óptico de simetría axial idealmente perfecta que transmite y refracta la luz haciéndola converger o divergir. Se basan en la ley de refracción de la luz, o ley de Snell, que a su vez se explica por el principio de tiempo mínimo de Fermat:

> *La trayectoria que sigue un rayo de luz entre dos puntos es aquella en la que emplea un tiempo mínimo en recorrerla.*

Precisamente son la adecuada forma simétrica, a lo largo del eje óptico, y el índice de refracción del material de fabricación los que provocan, alargando o acortando el camino que recorre la luz al atravesarla, que un frente de onda que incide plano termine convergiendo en un punto de la imagen, o por el contrario, diverja a partir de ahí. En este trabajo nos hemos interesado por las lentes convergentes como la mostrada en la figura 2.2.

El comportamiento descrito es el de una lente ideal. Aunque es prácticamente imposible de lograr con una sola lente, la ingeniería óptica ensambla

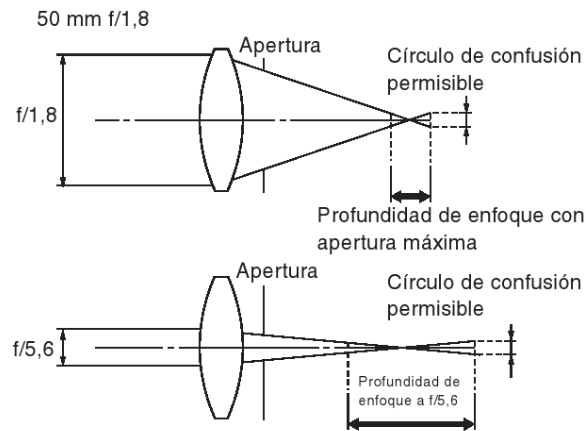


Figura 2.3: Relación entre apertura y profundidad de enfoque.

grupos de lentes cuyo comportamiento en conjunto se aproxima al ideal, y a pesar de su complejidad interna pueden ser descritas con unos pocos parámetros como son la longitud focal, el índice de apertura, los puntos principales, etcétera. Las definiciones al respecto se puede encontrar en cualquier libro que aborde la óptica, en especial la geométrica, por ejemplo [Hetch 01]. Para una visión práctica suele ser más conveniente usar un manual de fotografía como por ejemplo [London 04] o [Can 04]. Tan solo, revisaremos dos conceptos de especial significación para el desarrollo de este trabajo: la profundidad de campo y la formación de imagen

Profundidad de campo

Área que se encuentra delante y detrás de un motivo enfocado en la que la imagen fotografiada aparece nítida. Dicho de otro modo, la profundidad de la nitidez hasta la parte delantera y posterior del motivo donde el desenfoque de imagen en el plano focal se encuentra dentro de los límites del círculo de confusión permisible. La profundidad de campo varía en función de la longitud focal del objetivo, el valor de apertura y la distancia de disparo. La *profundidad de campo* se obtiene como:

$$\text{Profundidad de campo delantera} = \frac{d\,F\,a^2}{f^2 + d\,F\,a}$$

$$\text{Profundidad de campo posterior} = \frac{d\,F\,a^2}{f^2 - d\,F\,a},$$

donde f: longitud focal; F: número F; d: diámetro del círculo de confusión mínimo; y a: distancia desde el primer punto principal hasta el motivo.

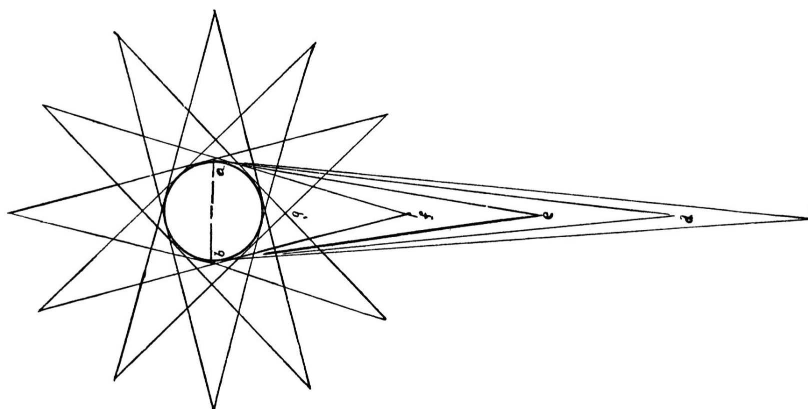


Figura 2.4: Pirámides de visión.

Formación de imagen La imagen que se forma en el interior de una cámara es proporcional a la irradiancia, que equivale a una integral pesada de la radiancia que atraviesa la lente. Sea $L_F(s,t,u,v)$ la radiancia del rayo que viaja de la posición (u,v) del plano de la lente a la posición (s,t) del plano del sensor; F la separación entre la lente y la superficie fotorreceptora; y ϕ el ángulo entre el rayo (s,t,u,v) y la normal al plano del sensor. Entonces la irradiancia en la posición (s,t) del sensor es

$$E_F(s,t) = \iint \frac{1}{F^2} L_F(s,t,u,v) cos^4(\phi) \mathrm{d}u\mathrm{d}v. \tag{2.1}$$

2.1.1. Función plenóptica

Previamente habíamos avanzado las palabras de Adelson y Bergen [Adelson 91] para introducir el concepto de *función plenóptica*. Cabe considerar también lo que Leonardo da Vinci llamó «pirámides de visión», y que se muestran en la figura 2.4, con las que anticipa su teoría de la perspectiva lineal [Richter 80]:

> *«Every body in light and shade fills the surrounding air with infinite images of itself; and these, by infinite pyramids diffused in the air, represent this body throughout space and on every side. Each pyramid that is composed of a long assemblage of rays includes within itself an infinite number of pyramids and each has the same power as all, and all as each. A circle of equidistant pyramids of vision will give to their object angles of equal size; and an eye at each point will see the object of the same size.»*

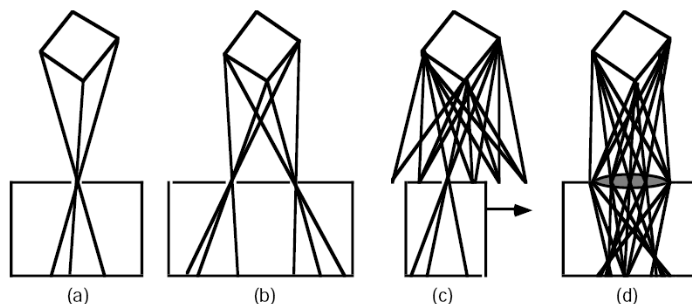


Figura 2.5: Varios esquemas básicos para capturar la función plenóptica. a) Cámara estenopeica; b) Par estéreo estenopeico; c) Paralaje por movimiento; d) Cámara con lente.

Ambos textos inciden en la misma idea de «cuanto puede ser visto», que no otra cosa significa *«plenoptica»*. Luz proveniente de todos los puntos de las superficies de los objetos, emitida en todas direcciones y ocupando todo el espacio: esto es, la totalidad de rayos en la escena.

Se puede estimar cuánta es esa información, comenzando por la imagen más simple: una cámara fotográfica estenopeica monocromática ubicada en un cierto punto. La imagen sensada daría idea de un solo punto de vista, en un único instante y promediada sobre todas las longitudes de onda del visible, esto es, $P(x, y)$, donde P es la intensidad de luz en el punto de coordenadas x, y, lo que corresponde a un campo de visión limitado por las coordenadas paramétricas. O más genéricamente $P(\theta, \phi)$, de modo que con dos ángulos se puede alcanzar toda la esfera que circunscribe al punto de observación. Si a esto se le añade el color se obtiene $P(\theta, \phi, \lambda)$. Que se convierte en una secuencia al añadir el tiempo: $P(\theta, \phi, \lambda, t)$. Por último, la totalidad de información sensable desde cualquier punto del espacio se describe por medio de $P(p_x, p_y, p_z, \theta, \phi, \lambda, t)$.

Esta función 7D es el objeto de estudio en la representación de escenas a partir de imágenes, *«Image Based Rendering»*. De hecho la *IBR* se puede enunciar como el problema de generar una representación continua de esa función a partir de un conjunto discreto de muestras de la misma. Es decir, cómo muestrarla y cómo reconstruirla.

En la figura 2.5, tomada de [Adelson 92], se muestran esquemas simplificados que posibilitan la captura de la función plenóptica. En los casos (a) y (b) de esa figura de esa figura se encuentran los modos de uso más frecuentes en los trabajos de visión, en los que, al suponer cámaras estenopeicas, en su configuración monocular o binocular, evitan el problema de la reconstrucción a partir de las muestras de la función. En complejidad ascendente, está (c)

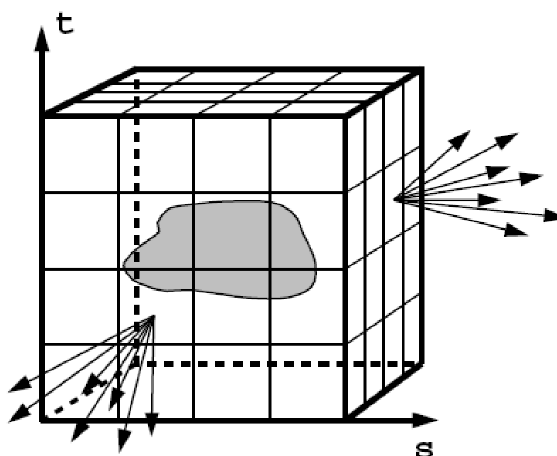


Figura 2.6: La radiancia de un objeto rodeado por un recinto 3D no cambia en el espacio libre exterior a éste.

la captura con movimiento perpendicular al eje óptico, que resulta en los *«Epipolar Plane Images»* estudiados en [Bolles 87], donde se usa el paralaje por movimiento para extraer la profundidad de los objetos. En (d) se pone de manifiesto que al abandonar la captura estenopeica, se tiene, se quiera o no, un muestreo más amplio de la función plenóptica, tal como si se integraran las vistas obtenidas en (c). En este caso se supone que el plano de enfoque recae sobre el vértice más cercano de la figura, con lo que su apariencia desde todas las vistas es similar y al integrarse su energía no se dispersa, apareciendo nítido, en cambio los otros vértices sufrirían desenfoque. En el trabajo de Adelson y Wang se incide sobre este punto para demostrar la viabilidad de aplicar técnicas de estereovisión a partir de una única cámara con una sola lente. Es el primer trabajo en muestrear explícitamente la función plenóptica en aras de obtener profundidades.

En las configuraciones vistas hasta ahora se están realizando las capturas sobre puntos de observación que están alineados o bien son coplanares y no sobre el volumen completo. Es una cuestión de simplicidad. Lo mismo que lo es el hecho de, sobre esas líneas o planos, adoptar un muestreo equiespaciado. Algunos autores dejan más libertad sobre la situación desde la que se observa la escena. Es el caso de [Buehler 01], donde no se impone ninguna restricción sobre las posiciones de observación.

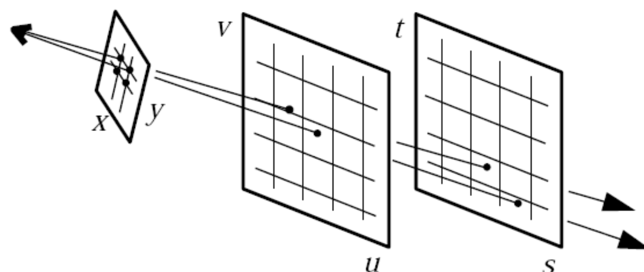


Figura 2.7: *Lightfield* parametrizado mediante 2 planos.

Parametrización discreta con 2 planos

Este último trabajo citado proviene del grupo de Gortler, Szeliski y otros, de Harvard y Microsoft Research, y son mejoras a su incursión pionera en este campo: el *«Lumigraph»* [Gortler 96]. Ese mismo año, y en la misma conferencia, es presentado por Levoy y Hanrahan, de Stanford, el concepto de *«Lightfield»*, [Levoy 96]. En uno y otro trabajo se retoman las ideas de Adelson y se simplifica el tratamiento de la función plenóptica al parametrizarla como una función 4D. Entiéndase que de partida se eliminan las componentes temporal y de color, por lo que restan 5 dimensiones, y solo queda por eliminar una de las dimensiones espacio-angulares. Esto se logra bajo la suposición de que el objeto, o el observador, se puede confinar en un cubo[3] fuera del cual no se permiten oclusiones, como se muestra en la figura 2.6. Si se permite situar cámaras libremente en ese espacio libre de oclusiones, éstas no aportarían más información de la que puede ser captada si estuvieran situadas sobre los planos que delimitan ese espacio. En la figura 2.7 el plano (s, t) es el mismo de la figura 2.6, esto es, corresponde a una de las 6 caras del cubo en que se confinaba el objeto. Por su parte (u, v) es el plano donde se situan las cámaras. Pasamos a hablar así de una función plenóptica 4D que contiene todo el flujo de luz entre ambas superficies. Estas cuatro dimensiones por lo general se expresan en coordenadas cartesianas, $P(s, t, u, v)$, al usarse una parametrización a partir de dos planos —*2PP: «two-plane parameterization»*—, aunque son posibles otras parametrizaciones [Gurrea 01]. Cuando en adelante nos refiramos a *lightfield* estaremos haciendo alusión a este tipo de función plenóptica.

En la figura 2.7 también se ilustra la posibilidad de recrear nuevas vistas alrededor del objeto, que es el objetivo de ambos trabajos. En la práctica es

[3] *Los términos específicos usados en la literatura son* «3D bounding box» *y* «convex hull».

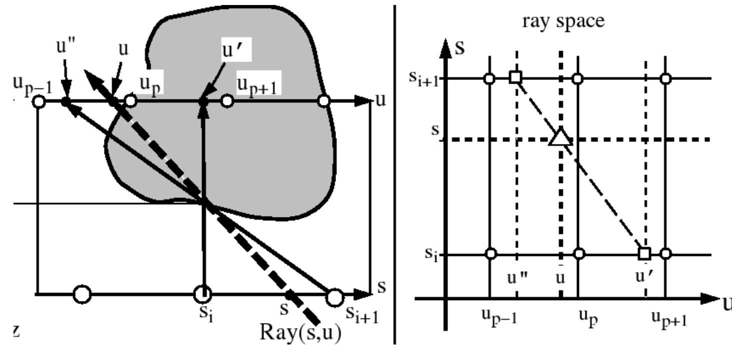


Figura 2.8: A la izquierda se observa en línea discontinua el rayo (s, u) que se desea crear por interpolación con los existentes (s_{i-1}, u_{p-1}), (s_{i-1}, u_p), ... (s_{i+1}, u_{p+2}). Si se conoce una aproximación de la geometría de la escena se pueden interpolar rayos más acordes, aunque no sean los vecinos más cercanos en el espacio de rayos, mostrado a la derecha.

necesario lidiar con un problema crucial: cómo interpolar los rayos $I(x, y)$ a partir de los contenidos en un *lightfield* discreto.

Las opciones usadas en [Gortler 96] y [Levoy 96] difieren. El *«Lumigraph»* hace uso adicionalmente de una aproximación geométrica del objeto. Con esto se propicia que para calcular un nuevo rayo se interpole, no ya de los que cortan ambos planos (s, t) y (u, v) cerca de donde lo hace el nuevo rayo, sino a partir de aquellos rayos que cortan con la superficie estimada del objeto cerca de donde se espera que lo haga el nuevo rayo, como se esquematiza en la figura 2.8. En el fondo, lo que supone es aceptar que el muestreo regular del plano (s, t), que proviene de la suposición inicial del confinamiento del objeto, desatiende las variaciones en profundidad del mismo y da lugar a que el muestreo sobre la superficie de éste sea irregular. Por lo tanto, de conocer, siquiera aproximadamente, la geometría del objeto se puede rebajar la densidad de muestreo de la función plenóptica en (s, t) y mantener la calidad visual de la reconstrucción. Pero se está condicionando la solución del problema a la bondad de la estimación de la geometría. El trabajo de Stanford, por su parte, ignora este hecho y por lo general aminora con sobremuestreo los artefactos que aparecen, además de proponer que el objeto coincida, en la medida de lo posible, con el plano teórico de confinamiento. Otros trabajos posteriores, como veremos seguidamente, desarrollan formalmente el tema del muestreo plenóptico.

Cabe considerar también, aún en el campo intuitivo, el concepto de «cámara de superficie», *Scam*, propuesto por Yu y McMillan del MIT, conjuntamente con Gortler, en [Yu 04], y que se ilustra en la figura 2.9. En ese

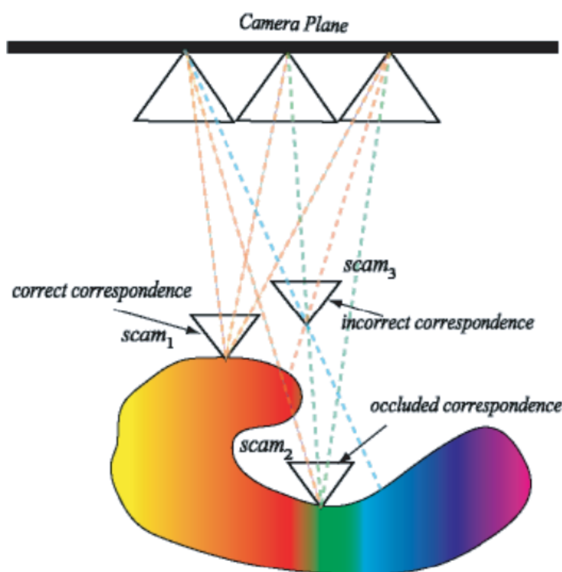


Figura 2.9: Cámara de superficie, *scam*.

trabajo se reparametrizan los rayos de la función plenóptica de tal manera que se agrupan aquéllos que se cruzan en un punto dado del espacio. Estas agrupaciones de rayos dan idea de la distribución angular de la radiancia, *«Bidirectional Reflectance Distribution Function», BRDF*, en esos puntos. Se puede considerar que, conocidas las BRDFs en torno a los puntos ubicados sobre el plano (s, t) —que no otra cosa es la función plenóptica 4D 2PP—, se pueden conocer las BRDFs de «todos» los puntos del espacio. Analizando la verosimilitud de estas BRDFs se puede estimar la forma de la escena, aunque para ello se ha de enfrentar un problema añadido: las oclusiones. De este trabajo a nosotros nos interesa resaltar la posibilidad de realizar esta reparametrización.

Dicho punto se aborda de manera independiente y más detallada en un trabajo bastante anterior, [Gu 97], del que nuevamente es coautor Gortler. El resultado más relevante de este trabajo es que en un *lightfield* el conjunto de rayos que se cruzan en un punto del espacio, (p_x, p_y, p_z), están contenidos en un subconjunto afín bidimensional, en un plano, descrito por:

$$plano(a, b) = (a, b, (1 + 1/p_z)a - p_x/p_z, (1 + 1/p_z)b - p_y/p_z). \quad (2.2)$$

En la figura 2.10 se plasma este resultado para el caso de trabajar en *«flatland»*: un mundo bidimensional que da lugar a una función plenóptica también bidimensional. En este caso, los rayos a traves del punto (x_0, z_0) de

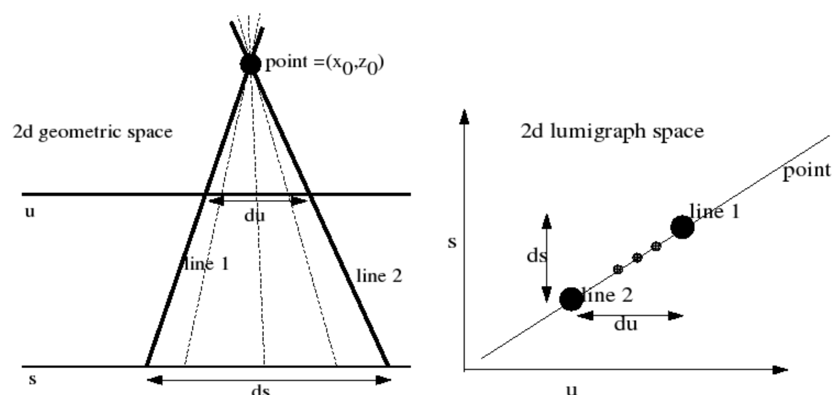


Figura 2.10: Rayos que se cruzan en un punto del espacio geométrico (x, z), se ubican sobre una línea en el espacio de rayos (s, u).

la geometría *flatland* coinciden en el espacio de rayos sobre una recta cuya pendiente $\frac{\mathrm{d}u}{\mathrm{d}s}$ está relacionada con la profundidad del punto, z_0.

Un trabajo que resulta significativo de cuánto se puede lograr simplemente reparametrizando la función plenóptica 4D es la *Master Thesis* de Isaksen en el MIT, [Isaksen 00]. Lejos de ceñirse a crear nuevas vistas, extiende su uso para lograr «efectos fotográficos» tales como apertura, enfoque y profundidad de campo variable, al modificar la manera en que se interpolan los nuevos rayos, esto es, reparametrizando y realizando integrales pesadas de la radiancia. Aunque es un compendio excelente de qué es posible hacer, sin embargo, no profundiza en los aspectos computacionales de los distintos métodos descritos, usando por lo general evaluaciones directas de esas integrales.

Por último este autor describe cómo mostrar estos *lightfields* mediante autoestereoscopía gracias a un array de microlentes que juega el papel inverso al que tendrá este elemento en los sensores plenópticos. Como ya se avanzó, por medios ópticos pasivos es posible captar y recrear una imagen tridimensional de una escena, a pesar del sobrecoste de procesar los distintos puntos de vista.

Hemos reseñado los avances registrados en el entorno de Gortler. El otro trabajo simiente del *light field rendering*, provenía de Stanford, del grupo de Marc Levoy. Ren Ng, bajo la dirección de este último, es autor de los trabajos más significativos para entender la presente Tesis: [Ng 05a], [Ng 05b] y [Ng 06]. En estos trabajos no solo se logra construir un sensor plenóptico en un solo cuerpo de cámara, sino que además se obtiene un algoritmo cuasilineal para el enfoque digital, que se verá en detalle en la sección 2.2. Veamos

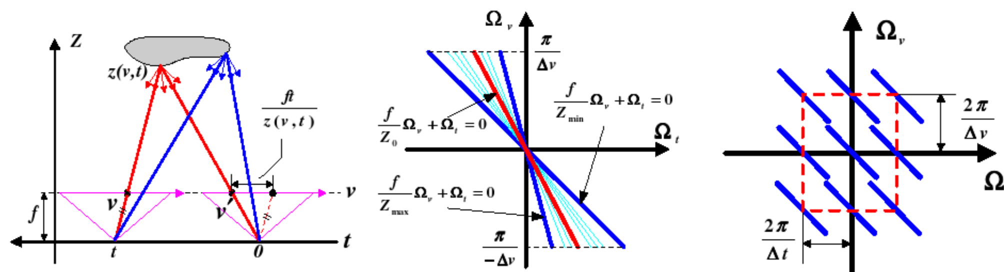


Figura 2.11: Soporte espectral de una escena de superficies lambertianas en *flatland*.

antes otras consideraciones respecto al muestreo de la función plenóptica.

2.1.2. Muestreo y sensores plenópticos

Los trabajos pioneros en *light field rendering* para contrarrestar los efectos de *aliasing* simplemente sobremuestreaban. Esto implicaba adquirir, almacenar y procesar más datos de los necesarios. A cambio, se eludia evaluar la tasa de muestreo adecuada, lo cual resulta complicado por la cantidad de elementos a considerar: la geometría de la escena, la textura de las superficies de los objetos, las propiedades reflectivas de esas superficies, el movimiento de los objetos, el número y posición de los puntos de observación, la resolución con que se muestrea cada vista, la posición de la(s) nueva(s) vista(s), la resolución de la(s) nueva(s) vista(s), y otros de menor relevancia.

Aunque teóricamente es posible resolver para todas las variantes consideradas y la función plenóptica 7D, en la práctica los distintos autores optaron por simplificar el problema.

En [Lin 00] se determina, mediante razonamientos geométricos, cuál es la distancia máxima entre las cámaras en la cual un único punto de la imagen aún no produce «doble imagen» al reconstruir mediante interpolación bilineal.

En [Chan 00] y [Chai 00] el estudio se realiza en el dominio de Fourier, determinando la tasa de muestreo a partir del ancho de banda del espectro del *lightfield*. Las conclusiones a las que se llega presuponen un comportamiento lambertiano en las BRDFs de los objetos, y que no existen oclusiones. Se ilustra en la figura 2.11. A la izquierda se aprecia el razonamiento geométrico que se emplea. En *flatland*, donde z es la profundidad, t la dimensión donde se ubican las cámaras y v el plano de llegada de los rayos ubicado a la distancia focal f, se tiene que $p(t, v) = p(0, v - \frac{f\,t}{z_0})$. De ahí se infiere que

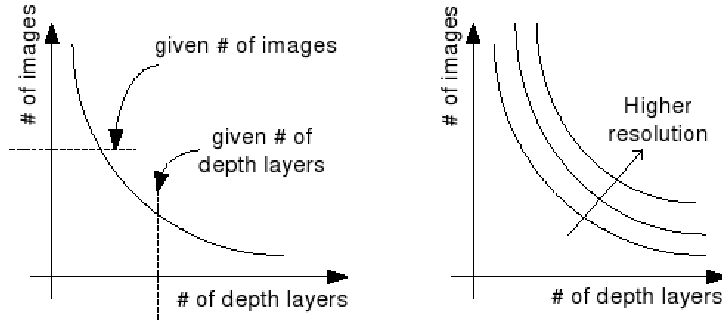


Figura 2.12: Curvas de tasa óptima de muestreo.

el soporte espectral es $P(\Omega_t, \Omega_v) = P'(\Omega_v)\delta(\frac{f}{z_0}\Omega_v + \Omega_t)$. Esto es, que si el objeto está a distancia z_0, y la radiancia de esa superficie no varía angularmente, el soporte espectral es una línea recta de pendiente $\frac{f}{z_0}$. Luego, será la relación entre la distancia focal y las distancias mínima y máxima de la escena las que delimiten el soporte total de su espectro —sin entrar a considerar cuán complicada es ésta: $P'(\Omega_v) = \mathfrak{F}\{p(0, v)\}$—, como se ilustra en la subfigura central. Conocidos los límites del soporte espectral se pueden fijar adecuadamente los periodos de muestreo (a la derecha). Adicionalmente se propone el filtro adecuado para compactar el soporte espectral, aunque para ello se ha de añadir información respecto de la conformación geométrica de la escena. Por último, en este trabajo se estudia la relación entre la fiabilidad de la aproximación geométrica de la escena y el número de imágenes que garantizan la ausencia de *aliasing*; y la relación entre la curva resultante y la resolución de las nuevas vistas. Estos resultados se recogen gráficamente en la figura 2.12.

Un análisis de mayor profundidad se puede encontrar en [Zhang 03a], [Zhang 03b] y [Zhang 04], donde se consideran superficies no lambertianas, muestreo no uniforme, y se proponen métodos de captura adaptativa.

Hemos visto que nuestros objetivos pasan por conservar la información angular de la radiancia. En la figura 2.13 se muestran sendos montajes que lo consiguen, pero haciendo uso de varias cámaras [Wilburn 02] y [Wilburn 05]. El montaje a la izquierda ubica coplanarmente un array de cámaras y las dirige a un plano focal común, de este modo se obtiene una función plenóptica 4D 2PP. El de la izquierda asume otra parametrización, pero es pertinente porque pone de manifiesto lo complejo de la instalación. Veremos que se puede capturar un *lightfield* de manera menos voluminosa.

Para lograrlo con un solo cuerpo hay que deshacer la presunción de que una cámara (no estenopeica) integrará la función plenóptica 4D sobre la aper-

Figura 2.13: Array de cámaras de Stanford

tura 2D produciendo así una proyección 2D. Hace ya más de un siglo que se propusieron modos de «no-hacer» esa integración: la *fotografía integral* [Ives 03], [Lippmann 08], [Roberts 03]. Más recientemente, diversos autores, [Adelson 92], [Ng 05b], [Georgiev 06], [Veeraraghavan 07], han vuelto sobre estos trabajos y los han llevado a la práctica en forma de prototipo. El resultado se ha dado en llamar *cámara plenóptica*, aunque según Georgiev sería más adecuado hablar de «fotografía de la radiancia». Este último autor establece un marco unificador de las distintas propuestas mediante la *óptica afín* [Georgiev 07a].

En esencia, lo que hay que conseguir es capturar individualmente los rayos que provienen de distintas direcciones. Como no se cambia el tipo de sensor: lo que incide sobre un mismo punto se integra, independientemente de la dirección de la que provino; lo que se ha de hacer es redirigir, hasta hacerlos incidir sobre distintos píxeles, los rayos provenientes de las distintas direcciones. En este sentido hay que jugar con la resolución espacio-angular de captura: solo se puede ganar en una a costa de la otra.

Cámaras estenopeicas

Es esclarecedor analizar cómo a partir de varias cámaras estenopeicas se puede construir el sensor plenóptico más simple. Véase al respecto la figura 2.14, donde en la parte superior izquierda se muestran los rayos caracterizados por la posición cartesiana de entrada y salida en los planos (s, t) y (u, v). Para las siguientes explicaciones es conveniente usar el esquema a su derecha, donde un rayo se caracteriza por una posición, q, y una pendiente, p, respecto al eje óptico.

La subfigura inferior izquierda representa la idea intuitiva de cámara obs-

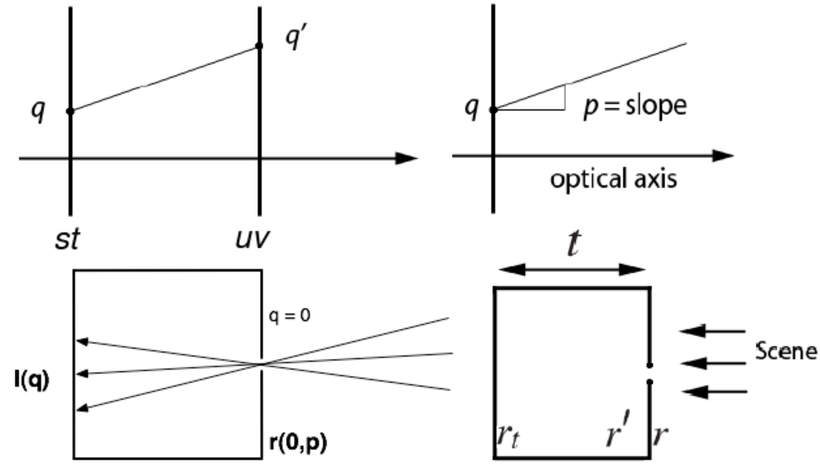


Figura 2.14: Análisis plenóptico de una cámara estenopeica.

cura: la radiancia de la escena $r(q, p)$, queda reducida en el estenopo a la porción que atraviesa el mismo, $r(0, p)$. Como se muestra a continuación, ayudados de la subfigura derecha, un sensor posicional ubicado en la pared interior opuesta al estenopo capturará la distribución angular de la radiancia: $I(q) \simeq r(0,\, q)$.

Para ello se distinguen tres radiancias: r antes de alcanzar el estenopo, r' tras atravesarlo y r_t al alcanzar la pared opuesta, tras viajar la distancia t sobre el eje óptico. Se tiene que $r'(q, p) = r(q, p)\, \delta(q)$. Esto es, el estenopo solo deja pasar infinitesimalmente la luz a su través, en la posición $q = 0$. Tras esto la luz se propaga en linea recta, con lo que al otro lado llega $r_t(q, p) = r(q - t\,p,\, p)\, \delta(q - t\,p)$. Por último, la imagen para cada posición, integrando angularmente, resulta

$$I(q) = \int r_t(q, p)\, \mathrm{d}p = \frac{1}{t} \int r(q - t\,p, \frac{t\,p}{t})\, \delta(q - t\,p)\, \mathrm{d}(t\,p) = \frac{1}{t}\, r(0, \frac{q}{t}).$$

Si ahora consideramos, en un mundo tridimensional, no una, sino varias de éstas, ubicadas contiguas sobre una rejilla, se obtendra un array 2D de imágenes 2D. Estamos ante un sensor plenóptico 4D. Y el principio de funcionamiento permite ubicar todas estas «microcámaras obscuras» dentro del cuerpo de una sola cámara, que tiene a su vez una única lente objetivo[4]. Éste

[4]Para que el sistema funcione correctamente la posición del estenopo debe variar ligeramente en función de la eccentricidad de cada microcámara, para que todas apunten al centro de la lente, y así el punto de corte sobre ésta de los rayos principales desde el pixel i-ésimo de cada cámara coincida. En la próxima subsección, cámaras con microlentes, sucede otro tanto, se ha de incluir una *«field lens»* adicional, con este mismo propósito.

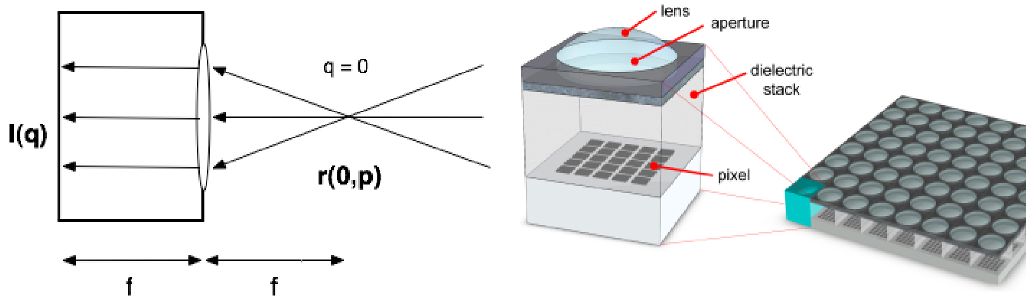


Figura 2.15: A la izquierda, esquema de funcionamiento de una microlente; a la derecha ensamblaje de varias microlentes sobre una CCD.

es el esquema propuesto por [Ives 03] y que inspira a [Adelson 92].

Cámaras con microlentes

El montaje con estenopos es de dificil manufactura y sufriría de difracción, por lo que es necesario incrementar esa apertura, y por lo tanto recurrir a lentes (denominadas «microlentes», por sus dimensiones y por diferenciarlas de la lente principal de la cámara). El principio de funcionamiento para su uso en sensores plenópticos se esquematiza en la figura 2.15.

Las microlentes se sitúan detrás de una lente objetivo con igual número F. En la subfigura de la izquiera se ilustra como los rayos que la lente principal enfoca a distancia focal f de las microlentes, éstas los separan posicionalmente, atendiendo a su ángulo de incidencia, sobre el sensor ubicado a distancia f tras ellas. Si se considera una apertura D, la integral de formación de imagen resultaría $I(q) = \frac{D}{f} r(0, \frac{q}{f})$. Esto es, el efecto es el mismo que en el caso estenopeico, pero se captaría D veces más luz.

A la derecha se muestra un esquema del ensamblaje de microlentes y sensor [Fife 06]. Las microlentes, junto con su sección de respaldo sobre el sensor, vendrían a jugar el papel de «macropíxeles» de una cámara plenóptica (los arrays 2D interiores de un array 2D externo). Este mismo principio de funcionamiento es el que rige en [Lippmann 08], de nuevo en [Adelson 92], y en [Ng 05b]. Aunque en este último trabajo se supone que el enfoque, y la separación de los rayos, se produce sobre la microlente, no a una distancia f por delante de ellas.

En un trabajo posterior, Lumsdaine y Georgiev afirman que no es necesario que el sensor esté ubicado a una distancia focal tras las microlentes [Lumsdaine 08]. En esa posición se maximiza la resolución angular obtenida, pero se puede sacrificar ésta y obtener mayor resolución espacial si se las aleja. Aún el *lightfield* capturado permitiría el reenfoque, aunque no con los

métodos que veremos a continuación.

También parte de Georgiev la propuesta de sustituir las microlentes por combinaciones de prismas y lentes, convergentes y/o divergentes, dando lugar a varios montajes equivalentes al propuesto: [Georgiev 06]. También se pueden utilizar sistemas ópticos que permitan posicionar las microlentes en el interior del objetivo, y reimaginarlas en el interior de la cámara, con lo que se posibilita la construcción de objetivos plenópticos intercambiables.

Sensores Shack-Hartmann

El uso de microlentes para intercambiar posición por distribución angular tiene un precedente en el montaje *Shack-Hartmann*, propuesto en 1900 por Hartmann, [Hartmann 00], y construido en 1971 por Shack y Platt, [Shack 71], y que es uno de los sensores de frente de onda más usado en el campo de la Óptica Adaptativa [Tyson 98]. Su función en ese ámbito es medir, indirectamente, la fase de la luz que alcanza la pupila de los telescopios situados dentro de la atmósfera terrestre, con el fin de caracterizar y corregir las aberraciones que ésta introduce y que de otra manera imposibilitaría el total aprovechamiento de la resolución de los mismos [Rosa 92] [Rodríguez-Ramos 97]. Su única diferencia respecto a las nuevas cámaras plenópticas es la ausencia de lente en la apertura, puesto que la luz en los telescopios proviene del infinito, y ya llega con frente de onda plano en ausencia de turbulencia atmosférica.

Cámaras heterodinas

Los montajes ópticos recién analizados logran sensar directamente la distribución angular del *lightfield* al intercambiar los papeles de posición y ángulo de incidencia. Hay otro tipo de sensores que podríamos denominar indirectos, puesto que codifican mediante máscaras la contribución angular del haz de rayos y aunque se produce la integración, es posible revertirla al decodificar las contribuciones individuales de cada uno. Nos resultan menos interesantes que las anteriores, porque añaden mayor complicación al procesado alejándolo del objetivo de tiempo real, si bien logran aprovechar óptimamente la superficie del sensor, pero se incluyen por completitud.

En esta linea van los trabajos de Veeraraghavan, Agrawal y Raskar del MERL: [Veeraraghavan 07], [Veeraraghavan 08]; y nuevamente Georgiev: [Georgiev 07b], [Georgiev 08]. Están basados en el teorema de la modulación en el dominio frecuencial 4D. Crean repeticiones espectrales del *lightfield* al hacerlo pasar por una máscara de patrones sinusoidales de alta frecuencia ubicada convenientemente entre lente y sensor. Luego, operando sobre la

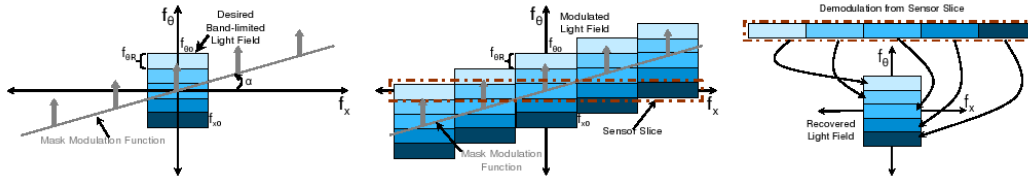


Figura 2.16: A la izquierda se aprecia el espectro limitado en banda de un *lightfield* continuo 2D (en *flatland*). Y cual será la función de modulación empleada: un tren de deltas inclinado y espaciado adecuadamente. En la subfigura central se aprecia el efecto de la modulación, y cómo las altas frecuencias espaciales del sensor (1D en *flatland*) están ocupadas por la información angular que de otra manera se hubiera perdido. A la derecha se muestra el proceso de reconstrucción del espectro del *lightfield* al reapilar adecuadamente.

transformada de Fourier de la señal sensada y reapilando las repeticiones espectrales, logran obtener el *lightfield*. Ver figura 2.16.

2.2. Volumen focal

Se puede seguir la traza a la idea de reenfoque *a posteriori*, por medios computacionales, en distintos trabajos: [Gu 97], [Isaksen 00], [Yu 04]. Era, sencillamente, una de las nuevas posibilidades que abría la fotografía computacional. Además su utilidad estaba lastrada por dos factores: el *aliasing* y el coste computacional.

Los reenfoques a partir de *lightfields* capturados con múltiples cámaras sufrían *aliasing* debido a la separación de las mismas. Esto se solventó, de manera teórica, al comprender adecuadamente los aspectos del muestreo plenóptico. Este problema además desaparece, en la práctica, con los montajes de sensores en un solo cuerpo, y los consiguientes *lightfields* «continuos».

Tampoco se prestaba inicialmente atención al método óptimo de computar este reenfoque. Se daba por hecho que en la integral y la interpolación requeridas primaría el coste de la primera y que sería un coste lineal con el tamaño del *lightfield*, pero sin posibilidad de reaprovechar cómputos en el caso de necesitarse más de un reenfoque a partir del mismo *lightfield*. Con lo cual, para un tamaño $N \times N \times N \times N$ y caso de desear obtener N planos «virtuales» de reenfoque, la complejidad sería como mínimo $O(N^5)$.

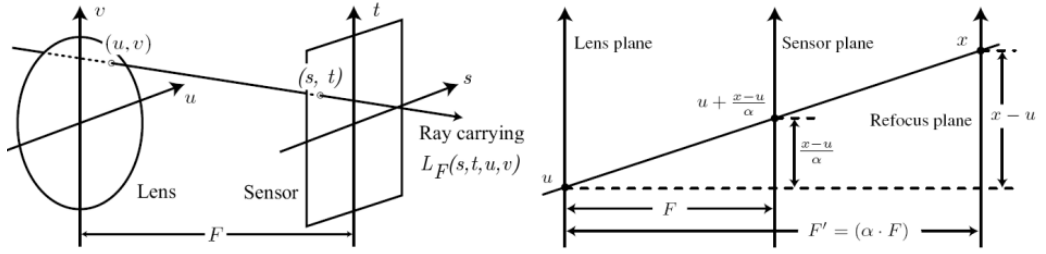


Figura 2.17: Reparametrización de los rayos de un *lightfield* para ubicarlos sobre un plano de enfoque virtual. A la izquierda, parámetros en la captura: donde (u, v) y (s, t) son, respectivamente, las coordenadas de corte sobre la lente y sobre el sensor, y éstos se distancian F. A la derecha disposición geométrica de los rayos que permite reenfocar sobre la distancia F'.

2.2.1. Reenfoque digital

En el trabajo de Gu, Gortler y Cohen [Gu 97], se identifica al conjunto de rayos que se cruzan en un punto del espacio. Esta misma relación, ecuación 2.2, la extrae por semejanza de triángulos Ren Ng en base a la figura 2.17.

Como consecuencia directa del transporte rectilíneo de la luz: el rayo que atraviesa (u, v) y luego corta en $(x-u, y-v)$ cuando se avanza $(1-\alpha)\,F$ sobre el eje óptico, hay que buscarlo sobre el plano del sensor en $(u+\frac{x-u}{\alpha}, v+\frac{y-v}{\alpha})$.

Luego, esa reparametrización tan sencilla nos permite conocer la radiancia sobre cualquier plano perpendicular al eje óptico. Y conocida ésta es posible recrear la imagen que se formaría en un sensor ubicado en dicha posición. Para ello se replica por métodos computacionales el proceso de formación de imagen, integrando angularmente la radiancia incidente sobre cada punto, según describe la ecuación 2.1. Esto es, se lleva a cabo finalmente el proceso de integración que el sensor plenóptico había pospuesto.

En total, se tiene que la nueva imagen sobre un plano a distancia $\alpha\,F$, en cada posición (x, y) del sensor vendría dada por

$$\mathcal{P}_\alpha[L_F](x,y) \simeq \iint L_F(u(1-\frac{1}{\alpha})+\frac{x}{\alpha},\ v(1-\frac{1}{\alpha})+\frac{y}{\alpha},\ u,v)\,\mathrm{d}u\,\mathrm{d}v. \quad (2.3)$$

Donde $\mathcal{P}_\alpha$ es, en la notación de Ren Ng, el operador de formación fotográfica y se han desestimado un factor constante $\frac{1}{\alpha^2 F^2}$, así como el decaimiento de la radiancia con el coseno elevado a la cuarta potencia del ángulo de incidencia, al no ser relevantes en términos del coste computacional.

Como se ha avanzado, la evaluación más simple de esta integral —hacerla equivalente a un sumatorio en el caso discreto y para una interpolación $O(1)$— ya supone un coste de $O(N^4)$, por plano.

Con vistas a extraer información geométrica a partir de él, deseamos construir el volumen continuo de planos reenfocados a lo largo del eje óptico. Es lo que denominaremos *volumen focal* de una escena, en el caso continuo. Reservaremos el término más común en la literatura *«focal stack»* para referirnos a la versión discreta en z de ese volumen focal. La complejidad de cómputo de un *«focal stack»* es función del número de planos que lo constituyen: siendo $O(N^5)$ para un volumen de N planos. Una implementación de esta complejidad, sobre GPUs, se encuentra en [Meng 07].

Esta complejidad se reduce con el método propuesto en [Ng 05a] a una complejidad cuasi-lineal, $O(N^4 \log_2 N)$, para computar hasta $O(N^2)$ planos.

2.2.2. Teorema de *slice* de Fourier

El método se basa en un principio que puede parecer alejado del tema tratado: el teorema de corte-proyección[5] en el dominio de Fourier.

Introducido en el campo de la radioastronomía por Bracewell [Bracewell 56], y clave en la tomografía computerizada, afirma que un corte unidimensional en el espectro de una señal bidimensional, representa la transformada de Fourier de la integral de esa señal proyectada en la dirección ortogonal a la dirección del corte.

Sea $f(x, y)$ una señal bidimensional, y $p(x)$ la proyección integral de $f(x, y)$ sobre el eje x,

$$p(x) = \int_{-\infty}^{\infty} f(x, y)\, \mathrm{d}y.$$

En ese caso la transformada de Fourier de la señal bidimensional es

$$F(k_x, k_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, e^{-2\pi i(xk_x + yk_y)}\, \mathrm{d}x\mathrm{d}y$$

El *slice* o corte unidimensional de ese espectro a lo largo del eje horizontal es $s(k_x) = F(k_x, 0)$, que coincide con la transformada de Fourier de $p(x)$

$$\begin{aligned} s(k_x) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, e^{-2\pi i x k_x}\, \mathrm{d}x\mathrm{d}y \\ &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(x, y)\, dy \right] e^{-2\pi i x k_x} \mathrm{d}x = \int_{-\infty}^{\infty} p(x)\, e^{-2\pi i x k_x} \mathrm{d}x. \end{aligned}$$

Esta propiedad, de sencilla demostración para la proyección sobre el eje x, es igualmente cierta para cualquier otro corte, y además se puede generalizar para más dimensiones. Es lo que hizo Ren Ng, dando pie al teorema generalizado de *slice* de Fourier, ilustrado en la subfigura izquierda en 2.18.

[5] *Fourier Projection-Slice Theorem*, en adelante lo denominaremos teorema de *slice* de Fourier.

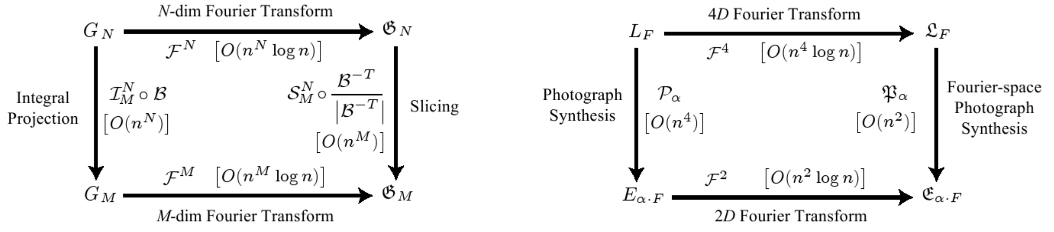


Figura 2.18: A la izquierda, el teorema generalizado del *slice* de Fourier. A la derecha, teorema de la formación fotográfica por *slice* de Fourier.

En su nomenclatura, $\mathcal{I}_M^N$ representa la proyección integral, que reduce una señal N-dimensional[6] a un espacio M-dimensional integrando sobre las últimas $N-M$ dimensiones:

$$\mathcal{I}_M^N[f](x_1, \ldots, x_M) = \int f(x_1, \ldots, x_N) \mathrm{d}x_{M+1} \ldots \mathrm{d}x_N.$$

$\mathcal{S}_M^N$ es el operador de *slice*, que anula las últimas $N-M$ dimensiones de una señal N-dimensional:

$$\mathcal{S}_M^N[f](x_1, \ldots, x_M) = f(x_1, \ldots, x_M, 0, \ldots, 0).$$

El operador $\mathcal{B}$, es de forma genérica, un cambio de base de los parámetros de la función. Ha de existir su inversa, que cumplirá $\mathcal{B}[f](\mathbf{x}) = f(\mathcal{B}^{-1}\,\mathbf{x})$, cuando se aplique al vector columna de parámetros $\mathbf{x}$. Con $\mathcal{B}^{-T}$ se representará a la traspuesta de su inversa.

$\mathcal{F}^N$ es el operador de transformada de Fourier en N dimensiones.

G_N y G_M representan a una función N-dimensional y su proyección integral a M dimensiones; mientras que $\mathfrak{G}^N$ y $\mathfrak{G}^M$, son sus respectivos espectros.

El teorema generalizado de *slice* de Fourier afirma que

$$\mathcal{F}^M \circ \mathcal{I}_M^N \circ \mathcal{B} \equiv \mathcal{S}_M^N \circ \frac{\mathcal{B}^{-T}}{|\mathcal{B}^{-T}|} \circ \mathcal{F}^N.$$

Esto es, si una señal f de dimensión N se reparametriza con el operador $\mathcal{B}$, se le calcula la proyección integral a M dimensiones y al resultado se le transforma al dominio de Fourier, se obtiene el equivalente a calcular primero la transformada N dimensional, cambiar a la base traspuesta normalizada de la original y por último realizar el *slice* a M dimensiones. Ambos caminos se

[6] Por coincidir con la notación usada por el autor, durante esta sección se usará n para designar el tamaño de la entrada, y N y M para la dimensionalidad de los datos antes y después de aplicar el slice

$$\begin{bmatrix} q' \\ p' \end{bmatrix} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix}$$

Figura 2.19: El transporte de luz equivale a una cizalla, o *shear*, de la radiancia.

ilustran en la subfigura izquierda 2.18, donde también se ha añadido el coste computacional de aplicar los distintos operadores.

Para llegar de G_N a G_M el coste es $O(n^N)$ aplicando directamente la proyección integral. Por el otro camino es ligeramente más costoso: implica $O(n^N \log n)$ para la transformada de Fourier sobre todas las dimensiones consideradas $+\ O(n^M)$ para realizar el corte que reduce la dimensionalidad $+\ O(n^M \log n)$ para la antitransformada desde $\mathfrak{G}_M$ hasta G_M.

Ahora bien, solo recorriendo cada camino una vez[7] resulta más costosa la última opción. Pero cuando se desea variar únicamente la reparametrización $\mathcal{B}$, en el primer camino se vuelven a necesitar $O(n^N)$ cómputos, mientras que en el otro, solo se ha de repetir la parte final: $O(n^M) + O(n^M \log n)$.

2.2.3. Fotografía por *slice* de Fourier

Podemos reconocer a los operadores recien definidos como los procesos que recoge la ecuación 2.3 de formación fotográfica sobre un plano a distancia αF. Así el proceso de formación de imagen sobre el sensor equivale a una proyección integral, mientras que trasladar el sensor es el equivalente a transportar la luz desde el plano de captura, en una operación geométrica de cizalla, ver figura 2.19, que queda descrita con el operador

$$\mathcal{B}_\alpha = \begin{bmatrix} \boldsymbol{\alpha} & \mathbf{1}-\boldsymbol{\alpha} \\ 0 & \mathbf{1} \end{bmatrix} \quad \mathcal{B}_\alpha^{-1} = \begin{bmatrix} \frac{\mathbf{1}}{\boldsymbol{\alpha}} & \mathbf{1}-\frac{\mathbf{1}}{\boldsymbol{\alpha}} \\ 0 & \mathbf{1} \end{bmatrix}$$

Esta matriz difiere ligeramente de la mostrada en la figura 2.19 porque la distribución angular aquí no viene expresada como derivada, sino por el punto de corte en cartesianas; pero la idea de fondo es la misma: se integrará sobre las coordenadas u y v; mientras que s y t varían acorde a u y v en la forma $s \simeq u\,\alpha + x$ y $t \simeq v\,\alpha + y$.

Por lo tanto, la operación de reenfoque de un *lightfield* se puede aprovechar del teorema generalizado de *slice* de Fourier, para el caso $N = 4$ y

[7]Para ser más precisos recorriéndolos menos de $\log_2 n$ veces.

$M = 2$, con solo usar el operador de *shear*, $\mathcal{B}_\alpha$, que hemos visto. En total,

$$\mathcal{P}_\alpha[L_F] \equiv \mathcal{I}_2^4 \circ \mathcal{B}_\alpha[L_F] \equiv \mathcal{F}^{-2} \circ \underbrace{\mathcal{S}_2^4 \circ \mathcal{B}_\alpha^{-T}}_{\mathfrak{B}_\alpha} \circ \mathcal{F}^4[L_F].$$

Se ilustra en la subfigura derecha 2.18. La alternativa a evaluar directamente la integral pasa por el dominio de Fourier:

1. Aplicar la transformada 4D de Fourier al *lightfield* L_F.
2. Por cada plano de reenfoque que se desee:
   a) Cortar $\mathfrak{B}_\alpha[\mathfrak{L}_F](k_x, k_y) = \mathfrak{L}_F(\alpha\, k_x, \alpha\, k_y, (1-\alpha)\, k_x, (1-\alpha)\, k_y)$.
   b) Antitrasformar 2D el corte para obtener la imagen $E_{\alpha F}(x, y)$.

Este proceso se ilustra en la figura 2.20, donde se aprecian, de izquierda a derecha:

- Un *lightfield*, en su conjunto y ampliado en las subimágenes de cada microlente. En las ampliaciones se pueden observar las dimensiones horizontal y vertical de microlente dentro de la imagen en su conjunto, y dimensiones horizontal y vertical de pixel dentro de una microlente.
- La transformada 4D de Fourier del *lightfield*, con ampliaciones donde se aprecian la distribución frecuencial en las 4 dimensiones.
- Tres posibles *slices* 2D de la función transformada 4D, y la antitransformación de esos cortes para obtener la imagen fotográfica correspondiente a distintos planos de enfoque.

Reenfoque digital

La teoría de la fotografía por *slice* de Fourier presupone señales continuas. Cuando se aplica el algoritmo anteriormente descrito sobre señales discretas surgen varios problemas en torno a cómo extraer un corte como el previsto en el paso 2.*a)* cuando dejamos de estar en el continuo. Habrá que interpolar dichos valores, con la particularidad de que la interpolación se da en el dominio transformado.

Dado que el filtro ideal es una *sinc* infinita y por lo tanto irrealizable, Ren Ng opta por usar un filtro de Kaiser-Bessel de longitud 2'5, y adicionalmente premultiplicar en el dominio espacial y sobremuestrear en el transformado, para contrarrestar los efectos de *roll-off* y *ghosting*.

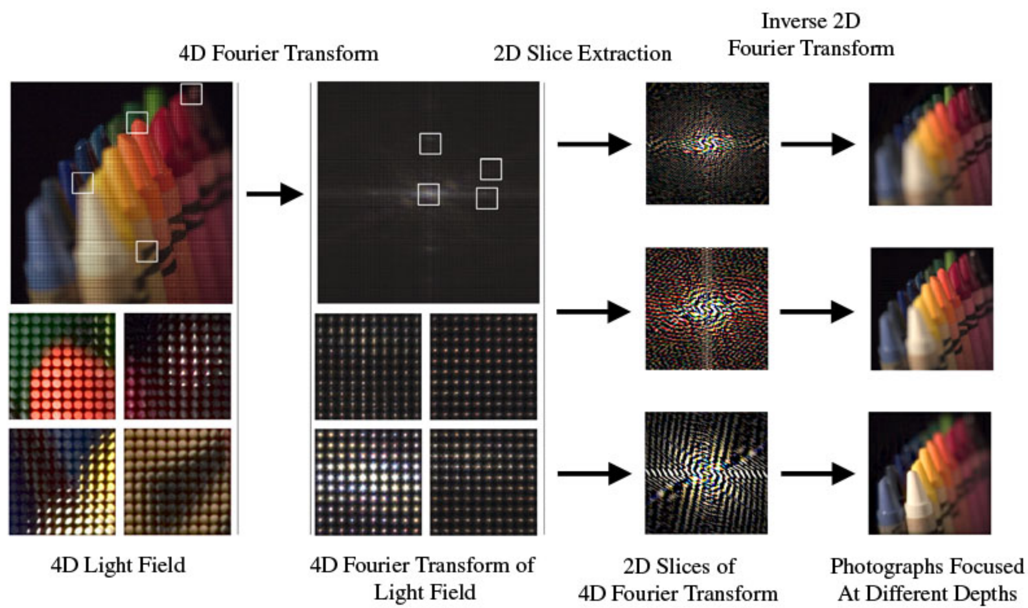


Figura 2.20: Fotografía por *slice* de Fourier. De izquierda a derecha: imagen procedente de un sensor plenóptico; su representación en el dominio transformado 4D de Fourier; y el operador *slice* 2D para obtener, previa antitransformación, la imagen reenfocada en distintos planos virtuales.

Discretización sobre una rejilla pseudo-polar Otro método basado en la generalización a 4D del teorema de *Slice* de Fourier para la evaluación de las integrales de formación fotográfica, pero que difiere del de Ren Ng en cómo obtener los valores discretos de los coeficientes sobre dichos *slices* se recoge en [Nava 08]. En este trabajo se realiza una interpolación trigonométrica 4D en el espacio de rayos y una discretización del operador integral basado en una generalización de la transformada discreta de Radon [Averbuch 08b].

La clave de esta propuesta es modificar la transformada de Fourier 4D inicial, desplazando las exponenciales complejas en una cantidad no entera, de tal manera que los coeficientes se obtengan directamente sobre una rejilla pseudo-polar [Averbuch 08a], en lugar de la rejilla cartesiana habitual, precisamente en aquellas combinaciones frecuenciales que conformarán los *slices* a antitransformar. De esta manera en lugar de interpolar los valores del espectro se opta por calcular de antemano aquéllos que nos van a interesar, trasladando la «fraccionalidad» desde las muestras hasta las exponenciales complejas, que sí se conocen en todo $\mathbb{C}$.

Para obtener un algoritmo rápido se expresa la transformada fraccional de Fourier [Bailey 91] —al igual que se hace con la *chirp-z* [Bluestein 70]— mediante una convolución, y es ésta la que a su vez se computa recurriendo a FFTs. Por ello la complejidad computacional —global— del algoritmo asociado a esta transformada está a la par con la que exhibe el algoritmo propuesto por Ren Ng —si bien es algo mayor por plano —.

Rendimiento El algoritmo propuesto por Ren Ng logra mejorar los tiempos obtenidos mediante evaluación directa de la integral de formación fotográfica. Aunque dependerá mucho del balance entre resolución espacial y angular, puesto que el ahorro del metodo de *slice* es mayor cuanto mayor sea la resolución en el plano angular (u, v). El autor da los siguientes tiempos, sobre un Pentium IV a 3 GHz y usando para el cálculo de la FFT la librería FFTW3:

Para un *lightfield* $256 \times 256 \times 16 \times 16$, mediante la evaluación directa, se consiguen 1'68 fps usando vecino más cercano como método de interpolación, y 0'13 fps usando interpolación cuadrilineal. Su método en cambio obtuvo 2'84 fps cuando uso un filtro de Kaiser-Bessel de longitud 1'5 sin sobremuestreo; y bajó a 0'54 fps con longitud 2'5 y sobremuestreo. Pero todo ello a costa de un preprocesado (el cómputo inicial de la transformada 4D) que consume 47 segundos.

Los resultados mejoraron hasta un orden de magnitud los de la evaluación de la integral para tamaño $128 \times 128 \times 32 \times 32$. En ese caso el método espacial logra 1.63 fps, cuando usa una interpolación simple, o 0.10 fps, si aplica una

mejor calidad en la interpolación. Frente a los 15'58 fps y 2'73 fps del método frecuencial, con un costo de arranque de 30 segundos.

Líneas abiertas por Ren Ng Ren Ng concluye su trabajo con los siguientes párrafos:

> *«A very different class of future work might emerge from looking at the footprint of photographs in the 4D Fourier transform of the light field. It is a direct consequence of the Fourier Slice Photography Theorem that the footprint of all full-aperture photographs lies on the following 3D manifold in the 4D Fourier space:*
>
> $$(\alpha\, k_x, \alpha\, k_y, (1{-}\alpha)\, k_x, (1{-}\alpha)\, k_y) \;\; \textit{where}\, \alpha \in [0,\infty), \;\; \textit{and}\, k_x, k_y \in \mathbb{R}$$
>
> *Two possible lines of research are as follows. First, it might be possible to optimize light field camera designs to provide greater fidelity on this manifold, at the expense of the vast remainder of the space that does not contribute to refocused photographs. One could also compress light fields for refocusing by storing only the data on the 3D manifold. Second, photographs focused at a particular depth will contain sharp details (hence high frequencies) only if an object exists at that depth. This observation suggests a simple Fourier Range Finding algorithm: search for regions of high spectral energy on the 3D manifold at a large distance from the origin. The rotation angle of these regions gives the depth (via the Fourier Slice Photography Theorem) of focal planes that intersect features in the visual world.»*

Usaremos ideas distintas a las que él propone, pero para aplicarlas precisamente con las dos finalidades que identifica: compresión y determinación de distancias.

2.2.4. Transformada de Radon de d-planos

Así pues, la fotografía por *slice* de Fourier — se use una transformada de Fourier 4D inicial en muestras cartesianas o pseudo-polares —, confían su velocidad a la eficiencia en el cómputo de la trasformada rápida de Fourier, FFT, que es el bloque computacional en que ambos métodos se sustentan. Este algoritmo, y su óptima implementación sobre la plataforma de las GPUs, será el tema de estudio en el capítulo 3. También cabe considerar si es posible obtener un algoritmo rápido sin recurrir a la FFT, y por lo tanto sin salir del dominio espacial. Nos adentraremos en esta cuestión observando la

trasformada de Radon de d-planos [Radon 17] puesto que, como hemos sugerido, las técnicas que conducen a encontrar algoritmos rápidos para dicha transformada pueden estar muy relacionados con la integral de formación fotográfica [Averbuch 08a, Averbuch 08b].

La transformada analítica de Radon, $\mathfrak{R}f(\theta,\rho)$, de una función bidimensional $f(x,y)$, $(x,y) \in \mathbb{R}^2$ calcula el conjunto de integrales de línea que se pueden describir por θ, el ángulo de las líneas respecto al eje y, y ρ, la distancia al origen. Esto es,

$$\mathfrak{R}f(\theta,\rho) = \iint f(x,y)\ \delta(x\ cos\ \theta + y\ sin\ \theta - \rho)\ \mathrm{d}x\ \mathrm{d}y, \tag{2.4}$$

Ésta es la forma en la que se suele definir en el campo de gráficos por ordenador, asociada a la transformada de Hough, donde es clave para el reconocimiento de formas [Ballard 82]. Ahora bien, el concepto original de transformada de Radon es más amplio. Y comprende el estudio de transformadas integrales de la forma

$$\hat{f}(\xi) = \int_{\xi} f(x)\,\mathrm{d}x.$$

Donde ξ es un hiperplano d-dimensional, $f \in \mathbb{R}^n$ y $\hat{f} \in \mathbb{R}^d$. Por lo tanto la transformada de Radon en el plano, que da cuenta de las integrales a lo largo de líneas, es el caso más sencillo: la transfomada 2-planar sobre un espacio 2-dimensional. También se la denomina *«ray»* o *«x-ray transform»*, por sus aplicaciones en imagen médica [Louis 83].

En una de las formas que adopta esta transfomada, reconocemos al operador fotográfico:

$$\mathfrak{R}f(\boldsymbol{s},\mathbf{d}) = \int f(\mathbf{u}\ \boldsymbol{s} + \mathbf{d},\ \mathbf{u})\ \mathrm{d}\mathbf{u} = \iint f(u_1\ s_1 + d_1,\ u_2\ s_2 + d_2,\ u_1,\ u_2)\ \mathrm{d}u_1\ \mathrm{d}u_2 \tag{2.5}$$

Nos interesará su estudio porque nos permitirá, en el capítulo 4, definir la *«transformada aproximada (rápida discreta) del* focal stack*», aFST*. Para ello hemos de extender el algoritmo que computa la transformada aproximada rápida discreta de Radon, que ya existía para el caso de integrales de línea en el plano, hasta llegar a integrales de ciertos planos sobre un hipercubo 4-dimensional. Con ello lograremos dar con algoritmos rápidos cuasi-lineales, e incluso uno lineal, que no requieren aritmética compleja, al no estar relacionados con los métodos de Fourier. Aplazaremos esta discusión hasta el capítulo correspondiente.

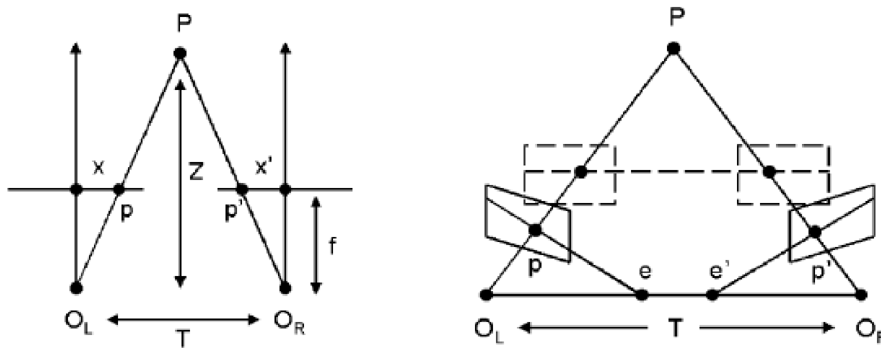


Figura 2.21: Geometría del problema de la estereovisión. A la izquierda, planteamiento básico para un par de vistas no vergentes. A la derecha, rectificación epipolar para convertir en líneas horizontales las zonas de búsqueda de emparejamientos.

2.3. Obtención de información tridimensional

La posibilidad de obtener información sobre la posición de un punto de una escena en el espacio tridimensional con técnicas pásivas de visión por computador se fundamenta en el hecho de que dicho punto generará imagen en un único par de coordenadas sobre dos cámaras que lo observan simultáneamente desde posiciones distintas, ver figura 2.21. Dadas las dos imágenes captadas por las cámaras, si es posible determinar la posición en que forma imagen el punto de la escena en cada una de ellas, x y x' en el esquema a la izquierda de la figura, es posible determinar por semejanza de triángulos la distancia sobre el eje óptico como $Z = fT/(x - x')$. Mientras este segundo paso, la *triangulación*, es inmediato, el determinar la *correspondencia* o *emparejamiento* de características es un problema ambiguo.

Es conveniente y común considerar el procedimiento de obtención de información geométrica de una escena compuesto por tres subtareas; las dos anteriormente citadas, más la *calibración*, que es el proceso por el que se determina la geometría externa del sistema de cámaras (sus posiciones relativas y orientación del punto de vista) e internas (longitud focal, centro óptico y distorsión de la lente). Estos parámetros son necesarios para relacionar las coordenadas expresadas en píxeles de las imágenes con las coordenadas externas de la escena. Si por el campo de aplicación, como en el caso que nos ocupa, prima conocer la relación en distancia entre los objetos sobre sus posiciones absolutas, esta subtarea no es estrictamente necesaria, bastará con conocer la disparidad, $x - x'$.

El conjunto de todas las disparidades entre las dos imágenes es lo que se conoce como mapa denso de disparidad. Habrá puntos en los que no se podrá

calcular la disparidad por no ser visibles desde ambas cámaras, al estar ocluidos. Puesto que la distancia se relaciona directamente con la disparidad por medio de los parámetros de calibración de las cámaras, se puede afirmar que el mapa de disparidad y oclusiones es equivalente al mapa de profundidades «2.5D» que se esbozó en la *Introducción*.

Cuando se dispone de dos imágenes calibradas, aún se las suele preprocesar en lo que se conoce como *rectificación*, usando las técnicas de la geometría epipolar [Hartley 04]. Esto permite proceder como si los ejes ópticos de las dos cámaras desde las que se tomaron las imágenes estuviésen alineados y la línea que une los centros de proyección de ambas cámaras, la línea de base del sistema, fuera paralela al eje horizontal del sistema de coordenadas. Entonces la búsqueda de correspondencias se reduce a evaluar una línea horizontal de barrido en las imágenes izquierda y derecha, tras hacer una transformación homográfica de las imágenes capturadas, según se esboza en el esquema derecho de la figura.

No es posible resolver satisfactoriamente el problema de las correspondencias entre píxeles de ambas imágenes cuando existan oclusiones, especularidades o falta de texturas, puesto que conduciran a soluciones ambiguas. Estos problemas desaconsejan emplear métodos de búsqueda locales de emparejamientos, aquéllos donde solo se tiene en cuenta el entorno cercano del pixel que está siendo evaluado, puesto que, a pesar de ser los más eficientes si se considera exclusivammente la velocidad de cómputo, son propensos a fallar en los casos mal condicionados anteriormente citados.

Por ello se recurre a métodos globales, más costosos computacionalmente pero menos sensibles a ambigüedades locales, donde las restricciones globales se expresan como un problema de minimización de energía, o modelos análogos, como la búsqueda del camino de flujo máximo en un grafo. Entre estos métodos se incluyen los de programación dinámica, corte de grafos, difusión y propagación de la confianza. La energía a minimizar se computa sobre una métrica tal como la correlación cruzada normalizada, la suma de diferencias al cuadrado o la suma de diferencias absolutas de los valores de intensidad de las imágenes.

Nosotros no realizaremos una investigación en profundidad en este campo, nos ceñiremos a usar uno de los métodos ya existentes, en concreto el de minimización de energía por propagación de confianza con paso de mensajes [Kolmogorov 05] aplicado sobre el volumen focal. Es posible aplicar ese o algún otro de los métodos habituales de estereovisión [Scharstein 02] pues las métricas que hagamos sobre el volumen focal están relacionadas con el cómputo de la correspondencia de los métodos de multiestereovisión [Schechner 00]. Por otro lado, las cámaras plenópticas no requieren de los pasos de calibración, ni rectificación estéreo. Tampoco del paso de triangulación

siempre que nos baste con hallar disparidades en lugar de distancias.

2.4. Hardware gráfico

Ren Ng con su algoritmo cuasi-lineal basado en la mejor implementación disponible de la FFT —la librería FFTW3 [Frigo 05]—sobre una única CPU —familia *intel x86* a 3 GHz— obtiene los siguientes tiempos: 47 segundos de preprocesado más 0'352 s (baja calidad) ó 1'852 s (alta calidad) por plano para tamaño $256^2 \times 16^2$, y, 30 s más 64 ms ó 366 ms por plano para tamaño $128^2 \times 32^2$.

Lo alejado de estos tiempos, los mejores en la literatura, de nuestro objetivo de tiempo real nos hace plantearnos la busqueda de alternativas, no solo al algoritmo de cómputo del *focal stack*, sino también al empleo de las CPUs como recurso computacional.

2.4.1. Paralelismo, arquitectura y paradigma de programación

Para obtener un *focal stack* a partir de un *lightfield* se han de procesar gran cantidad de instrucciones aritméticas con sus consiguientes accesos, de lectura y escritura, a memoria. El rendimiento que se consiga dependerá de lo rápido que se efectúen ambos: accesos y operaciones.

Las posibilidades de un programador de optimizar las implementaciones existentes en la literatura son reducidas si se emplea un solo procesador. En cambio, estas posibilidades aumenta en gran medida cuando se dispone de muchos procesadores. La optimización se consigue paralelizando el algoritmo, llevando a cabo varias de estas lecturas, operaciones y escrituras simultáneamente en las diversas unidades funcionales del recurso computacional.

Cómo llevar a cabo esta optimización por paralelización dependerá del problema y de la máquina dada. Del problema, en tanto que las dependencias entre resultados intermedios condicionan cuándo y cuántos de los cómputos se pueden efectuar en paralelo. Mientras que la la elección de la máquina, será la que fije el número de recursos y la manera de proceder para aprovecharlos al máximo.

El recurso computacional convencional, y que hemos descartado, son los microprocesadores que usan los ordenadores personales. Su arquitectura sigue el modelo original, de ejecución secuencial, de Von Neumann: *«instruction stream based»*, *ISB*, en la terminología de [Hartenstein 03]. Pese a la simplicidad de este modelo —un solo hilo de ejecución, una instrucción, ya sea aritmética o de lectura/escritura, a la vez— su rendimiento en términos de

flop/W no ha hecho sino aumentar desde que Moore predijo, en 1965, el crecimiento exponencial del número de transistores por unidad de superficie en los chips. Mientras se ha podido aumentar el número de transistores en un chip, mayor ha sido el rendimiento que se ha logrado con esta arquitectura.

Como en otros muchos problemas de ingeniería, un diseño inicial regular puede triunfar sobre otras alternativas, en base a razones de índole histórico o económico. Esta hegemonía «de tradición-coste» supone con el tiempo una hegemonía de «diseño», por el nivel de refinamiento que alcanza la elección inicial, y que, ahora sí, aguanta la comparación con los diseños alternativos, los cuales quedan reducidos a nichos del mercado donde el ratio rendimiento/coste aún les permite sobrevivir. En el caso que nos ocupa, estas mejoras incrementales se han concretado en la explotación de la localidad espacial y temporal en los accesos a memorias —por medio del uso de jerarquías de cache— y de la disponibilidad de múltiples unidades funcionales para un solo hilo de ejecución —por medio de ejecución especulativa y fuera de orden—. Dando así lugar a un modelo híbrido que mantiene al programador ajeno a las optimizaciones por paralelización que permite cada problema particular, mientras en su interior alberga un alto grado de paralelismo genérico.[8]

Sin embargo las arquitecturas ISB empiezan a perder esa hegemonía, puesto que ya no garantizan ni el mejor ratio flop/W, ni el mejor ratio flop/esfuerzo de programación [Sutter 05], [Dipert 05]. Las causas hay que buscarlas en el *«memory gap»*, la creciente distancia entre la capacidad bruta de computar, y la capacidad de alimentar las unidades de proceso con datos que hay que traer y llevar desde memoria. La gráfica 2.22, tomada de [Owens 05a], que a su vez la recopila del informe [ITRS 03], muestra las variables más importantes que afectan al rendimiento de los procesadores según la previsión de la propia industria de semiconductores, para la década 2004-2014, coincidiendo su inicio con el de esta Tesis. Mientras el consumo se mantiene estable, el resto de variables mejoran considerablemente. Pero, no todas en igual medida. Mientras que la proyección para ese periodo indica una reducción a la mitad de la latencia de memoria, variables como el ancho de banda a memoria o el número de transistores se multiplican por 10.

Esto tiene importantes repercusiones en una arquitectura que ha de incluir, por cada nueva de unidad de cómputo, más espacio de cache y lógica de control. Además ese aumento debe ser exponencial para poderse apreciar un incremento lineal del rendimiento. Esta tendencia es la que ha propiciado

[8]Esto es una simplificación, por supuesto, los microprocesadores secuenciales, dan la posibilidad de explicitar el paralelismo a distintos niveles, ya sea usando conjuntos de instrucciones SIMD específicas para multimedia, bien sea mediante SMP o multithreading, o, a otro nivel, mediante su agrupación en clústeres o, llevado al extremo, mediante programación distribuida.

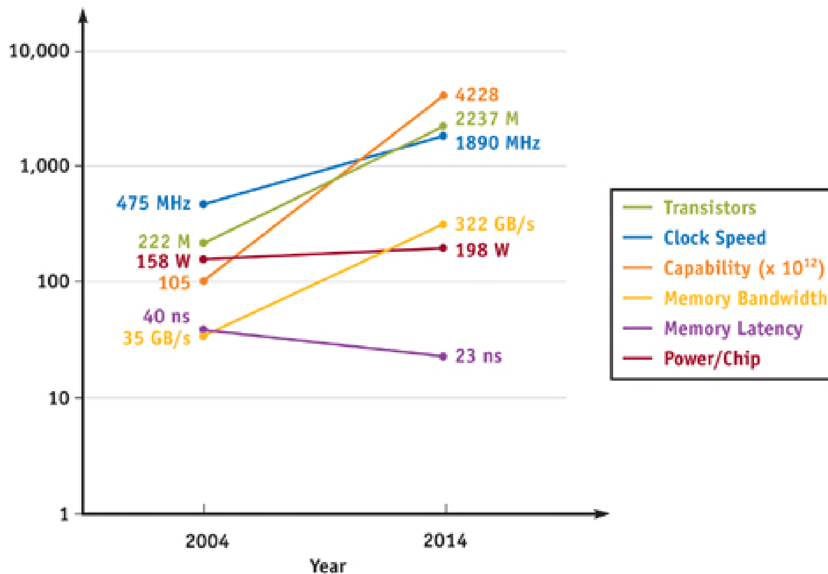


Figura 2.22: Evolución prevista del número de transistores, velocidad de reloj, potencia de cómputo, ancho de banda de memoria, latencia y consumo durante el periodo 2004-2014 según el [ITRS 03].

que, a día de hoy, la superficie dedicada a cómputo no suponga más que una pequeña fracción en muchos microprocesadores.

1 billón de transistores

En 1997 un número especial de *IEEE Computer* [Patt 97] pronosticaba cómo se afrontaría el diseño de microprocesadores con la futura disponibilidad de más de 1 billón de transistores[9]:

> Billion-transistor processors will be much as they are today, just bigger, faster and wider. [...] The current uniprocessor model can provide sufficient performance and use a billion transistors effectively without changing the programming model or discarding software compatibility.

[9]En 2006 Intel rompió la barrera del billón de transistores. Para finales de 2008 anuncia un chip con 2 billones: un *quad-core* con 30MB de memoria cache. Cosas del azar, en la presentación del mismo, como ejemplo de nuevos campos de aplicación que abriría, se presentó el trabajo de Ren Ng [Kaplan 08]. AMD-ATI y NVIDIA, los dos mayores fabricantes de GPU han lanzado comercialmente en 2008 chips que rebasan la cifra del billón de transistores, a pesar de usar tecnologías de fabricación menos sofisticadas que Intel.

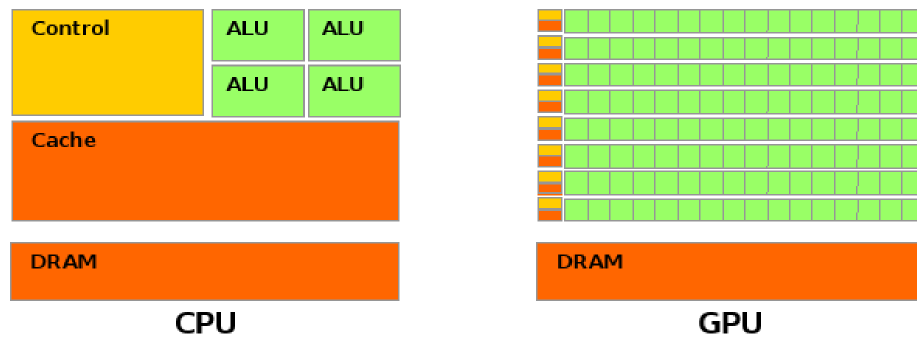


Figura 2.23: En verde, espacio dedicado a cómputo en una CPU y en una GPU.

En 2004 volverían sobre las predicciones hechas tan solo siete años antes [Burger 04]. Las conclusiones fueron las siguientes:

> Of seven architectural visions proposed in 1997, none has yet emerged as dominant. However, as we approach a microarchitectural bound on clock speed, the primary source of improved performance must come from increased concurrency. Future billion-transistor architectures will be judged by how efficiently they support distributed hardware without placing intractable demands on programmers.

Esto es, llegado un punto, para aprovechar plenamente los recursos que brinda la evolución en la tecnología de fabricación de chips es más conveniente exponer explícitamente el paralelismo intrínseco a los distintos algoritmos que esperar a que lo descubran y aprovechen los mecanismos genéricos implementados a nivel de compilador o microarquitectura en el modelo ISB. La alternativa ha de incluir, junto con la arquitectura hardware adecuada, las herramientas que permiten programarla de manera efectiva.

Esto es lo que ha sucedido con una amplia gama de problemas pertenecientes al ámbito del cómputo científico, y el binomio hardware/software que sobrevivía en el nicho de los gráficos 3D por ordenador.

La idea más radical de las GPUs es de una sencillez diáfana: dedicar más espacio al cómputo, a costa de detraerlo de caches y control, como se ilustra en la figura 2.23. Devestidos de estos soportes, propios del modelo ISB, se puede disponer de cientos de unidades de procesamiento sencillas en ese billón de transistores. Cuando la carga de cómputo se reparte en paralelo sobre centenas de núcleos, en lugar de sobre unos pocos —2, 4 u 8—, los accesos a memoria se solapan en el tiempo, se aumenta el *troughput* y se enmascara el problema de la latencia. Los aumentos de rendimiento no

están tan condicionados a la capacidad de seguir aumentando la velocidad del procesador, y la potencia se disipa de manera más homogénea en la superficie del chip evitando puntos calientes.

Un modelo tal requiere un sacrificio en la generalidad del procesador, con lo que no todo tipo de ámbitos de la computación pueden aprovechar al máximo esta arquitectura y su paradigma de programación asociado. Las GPUs, que se desarrollaron y sobrevivieron como un coprocesador específico, ahora que aspiran a extenderse fuera de su ámbito, aún lo hacen restringiéndose a un campo de aplicación reducido: servir de co-procesador de «supercomputación» a la CPU. O como se le ha denominado en alguna ocasión: *«stream coprocessor»* [Macedonia 03], [Owens 05a], al erigirse como el representante más extendido de las arquitecturas que [Hartenstein 03] denomina *«data stream based», DSB.*

Y se denominará también *«stream processing»* al paradigma de programación que surge[10] naturalmente de este tipo de arquitectura centradas en el paralelismo a nivel de datos: por cada dato hay que ejecutar un programa muy similar; los cómputos sobre cada dato son en buena medida independientes de los realizados sobre los otros datos; y el número de estos datos-programas es tan grande como para poder aprovecharse de la existencia de cientos de procesadores. Este tipo de problemas aparece continuamente en el campo de procesado de señales, y fue para las aplicaciones de ese campo que se construyeron los primeros *«stream processors»*: *Imagine* [Khailany 01], y *Merrimac* [Dally 03]. Parte de los desarrolladores de los mismos están vinculados a la evolución vivida recientemente por las GPUs [Buck 03a], como también lo está el lenguaje *Brook* [Buck 03b] creado para programar Merrimac.

2.4.2. Evolución de las GPUs

Las actuales GPUs tienen una historia breve. No existían antes de 1999, o 2001, dependiendo de cómo se acote.

La primera generación de tarjetas gráficas, previas a 1987, no ofrecía potencia de cómputo para ir más alla del renderizado de los modelos de alambre (*wireframe*); entre 1987 y 1992, se muestran sólidos básicos, sombreados con las técnicas Flat, Phong y Gouraud; con el mapeo de texturas, 1992–2000, se siembra el campo para llegar a donde estamos hoy; efectos programables, a partir de 2001; pero sin alcanzar aún los efectos de iluminación global[11].

[10] En la historia de la computación se puede establecer más que similitudes entre las actuales GPUs y las *Connection Machines*, [Hillis 86], y entre los lenguajes de programación de la CM-5, el C* [TM Corp. 93], y el actual CUDA [NVIDIA Corp. 07].

[11] Aunque estamos a las puertas. NVIDIA recientemente ha mostrado un trazador de rayos, con algunas limitaciones, que alcanza 30 fps sobre 4 GPUs [Luebke 08].

Tarjeta	Fecha	MTris/s	Gpíx/s	GBytes/s	GHz	Pipelines
TNT	Feb-99	6				
Voodoo3	Abr-99	6				
TNT2	Sep-99	9				
GeForce 256	Feb-00	15				
Radeon	Jul-00	30				
NVIDIA GeForce3 (NV20)	Mar-01	40	0.8	7.36	200	4
Radeon 8500	Aug-01	63	1	9	275	4
NVIDIA GeForce4 (NV25)	Feb-02	136	1.2	10.3	300	4
Radeon 9700 Pro	Jul-02	325	2.6	19.8	325	8
GeForceFX (NV30)	Feb-03	350	2.0	16	500	4
Radeon 9800 XT	Sep-03	412	3.3	23.4	412	8
GeForceFX 5950 (NV38)	Oct-03	356	1.9	30.4	475	4
GeForce 6800 (NV40)	Abr-04	600	6.4	35.2	400	16
ATI X800 XT	May-04	780	8.3	35.8	520	16
GeForce 7800 (G70)	Jun-05	-	6.9	40	430	24
“ 7950 GX2 (2xG71)	Jun-06	-	16	76.8	500	2x24
“ 8800 GTX (G80)	Nov-06	-	13.8	86.4	575	128
“ 8800 Ultra (G80)	May-07	-	14.7	103.7	612	128
“ 9800 GX2 (2xG92)	Mar-08	-	2x9.6	128	600	2x128
“ 9800 GTX (G92)	Apr-08	-	10.8	70.4	675	128
“ GTX 280 (GT200)	Jun-08	-	19.3	141.7	602	240

Cuadro 2.1: Medidas del rendimiento de las GPUs previas al inicio de esta Tesis: 1999-2004. Y exclusivamente de los modelos de NVIDIA, desde esa fecha hasta 2008. Los valores en las columnas corresponden a: capacidad para transformar polígonos, en millones de triángulos por segundo; capacidad para colorear/dar salida a pantalla, en gigapíxeles por segundo; ancho de banda de memoria; frecuencia de procesador; y unidades de procesado independiente. La primera línea horizontal, entre 2000 y 2001, marca el comienzo de la programabilidad, muy básica al principio, de las GPUs. Desde entonces, hasta 2004, la capacidad de procesado, en término de capacidad de transformación geométrica se incremento en un 600 % anual, y, de relleno de píxeles, en un 300 %. Por encima de la predicción de Moore. Tras la segunda línea horizontal se recogen los datos propios de los modelos comercializados por NVIDIA durante la realización de esta Tesis.

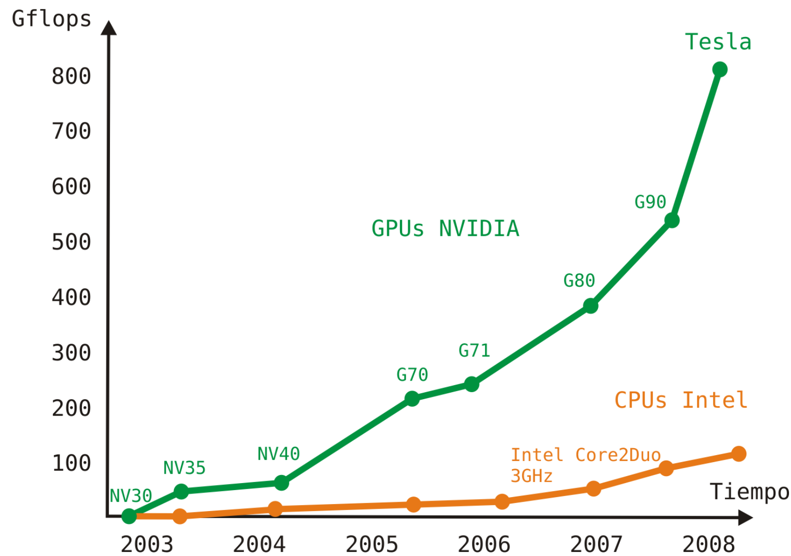


Figura 2.24: Evolución del rendimiento de CPUs y GPUs.

En la tabla 2.4.2 se recogen las características básicas, en cuanto a rendimiento, de los productos comercializados entre el inicio de la tecnología en 1999, hasta 2008, marcando el inicio de la programabilidad, en 2001, y el inicio de esta Tesis, en 2004.

La evolución de las mismas en tan corto periodo de tiempo —incluso los artículos germinales de Olano [Olano 98] y Lindholm [Lindholm 01] no distan más de una década del presente momento— se aprecia en el curso de la Universidad de Stanford, disponible en la red *Internet* en el momento de edición de este documento, sobre arquitecturas para el procesado de gráficos en tiempo real, impartido en 2001 [Akeley 01], y su reedición en 2007 [Akeley 07].

En la gráfica 2.4.2 se observa el despegue en rendimiento de las GPUs, respecto al exhibido por las CPUs coetáneas, y cómo mientras que un modelo parece resignarse a crecer linealmente, el otro podría aprovechar la tendencia exponencial de Moore en el número de transistores para extraer incrementos de rendimiento también exponenciales. Esa es la causa. La consecuencia es que en el «informe del estado del arte» del Eurographics'05 [Owens 05b], se recogen hasta 100 entradas bibliográficas, lo que da cuenta de la súbita madurez de un campo que la comunidad científica no ha dudado en tratar de aprovechar.

2.4.3. Programabilidad

No es posible en cambio sintetizar en una gráfica los avances producidos en el apartado de programabilidad de la plataforma.

Antes de 2001, el proceso para mostrar una escena en pantalla consistía en una serie de pasos prefijados que admitía pocos cambios, que en todo caso estaban también prefijados. El programador contaba con APIs específicas, como OpenGL [Segal 03] y DirectX [Microsoft 98] para acceder al *pipeline* gráfico. Que era una mezcla entre fijo y «configurable». Pero en ningún caso reprogramable.

Tímidamente, a partir de ese momento se abrió la posibilidad de realizar cambios al procesado que sufrían vértices y fragmentos —nombre que reciben los píxeles mientras se procesan, antes de llegar a pantalla— de una escena hasta convertirse en una imagen de la misma. Esta programabilidad era a bajo nivel y venía especificada, para OpenGL[12], en las extensiones al estándar [OpenGL ARB 01], [OpenGL ARB 02].

La programabilidad a alto nivel llegaría pronto con los lenguajes *GLSL*, del conjunto de la industria, [OpenGL ARB 03]; *Cg* de NVIDIA [Fernando 03]; y *HLSL* de Microsoft [Gray 03]; codiseñados simultáneamente y con diferencias mínimas entre ellos. Aunque se incluían cada vez herramientas más propias de lenguajes genéricos, como el soporte de coma flotante, y la posibilidad de ejecutar búcles, estos lenguajes seguían estando muy ligados a la programación gráfica. Y resultaba forzado hacer uso de ellos en otros ámbitos.

El término *«GPGPU»*, acrónimo en inglés de computación de propósito genérico sobre GPUs, fue acuñado en 2003 por Harris [Harris 03][13]. Tanto en esa tesis, como en otras del mismo periodo, como [Purcell 04], se comienza a hacer un uso extensivo de las GPUs como recurso computacional, y se preconiza la adopción de un nuevo paradigma de computación que sirva de marco para la adaptación y evaluación de algoritmos en esta plataforma. Este esfuerzo de formalización ha dado frutos en metalenguajes de programación como *BrookGPU* [Buck 04], que permiten conceptualizar la GPU como un *«stream processor»*, finalmente desligado de la programación gráfica.

Otros lenguajes de segunda generación, aparte del reseñado *BrookGPU* [14], son *SHader algebra*, de la Universidad de Waterloo [McCool 04], y su

[12]En DirectX se introduce un lenguaje ensamblador al respecto en la versión 8, en 2000. Los avances en esta rama no se tratan en este trabajo.

[13]Ese acrónimo da nombre al foro de desarrolladores más importante sobre esta plataforma. Es un repositorio de artículos, noticias, cursos y conferencias donde se puede rastrear la evolución de las GPUs en su uso con fines científicos [GPGPU 02].

[14]Descendiente del lenguaje *Brook* que servía para programar los *«stream processors»*

evolución comercial *Rapid Mind* [Monteyne 08]; o la alternativa de Microsoft, *Accelerator* [Tarditi 06].

Todos ellos no dejan de ser metalenguajes que permiten a programadores de otras áreas aprovechar los recursos de las GPUs escondiéndoles las interioridades de los lenguajes gráficos.

Esta espiral de aumento de generalidad, y menos ligazón con las especificidades del ámbito de gráficos por ordenador culmina, actualmente, en el apartado hardware, con GPUs totalmente orientadas a la *high-performance computing*, con las soluciones *Tesla* [Lindholm 08], de nVidia, y *FireStream* [AMD-ATI 08] de AMD-ATI[15]; y, en el apartado software, con los entornos de desarrollo *CUDA* [NVIDIA Corp. 07], por NVIDIA, y *Stream SDK* [AMD-ATI 07], por AMD-ATI[16].

Además de las iniciativas de los dos mayores fabricantes de GPUs son reseñables también: *OpenCL* [OpenGL-Apple 08] llamada a ser la extensión orientada a computación de OpenGL (impulsada por Apple); *Ct* un lenguaje para multi-cores de Intel [Ghuloum 07]; y *Larrabee* anunciada como la primera GPU del fabricante Intel, prevista inicialmente para 2009 [Seiler 08], pero parece que finalmente descartada.

2.5. Sumario

En este capítulo se han visto los modos de realización de sensores plenópticos para capturar el *lightfield* de una escena. Se han descrito dos métodos que, basados en el uso de la transformada rápida de Fourier, consiguen computar el *focal stack* de una escena a partir de su *lightfield*, con una complejidad computacional cuasi-lineal. Hemos apuntado a la posibilidad de rebajar esta complejidad, hasta hacerla lineal, extendiendo la transfomada de Radon para computar la suma a lo largo de ciertos planos en los *lightfields* entendidos como hipercubos 4D discretos. También hemos citado una técnica que permite calcular el mapa de profundidades más probable a partir de un *focal stack*.

Se han repasado las características y evolución de las GPUs con vistas a emplearlas como recurso computacional que permita acercar los tiempos de cómputo de los distintos algoritmos involucrados a los requerimientos de tiempo fijados por el objetivo de su visualización *online* en displays autoestereoscópicos de formato *2D + depth*.

De los distintos lenguajes existentes durante la realización de la Tesis

pioneros, diseñados en Stanford [Dally 03], [Khailany 01].

[15] En 2007 el fabricante de microprocesadores AMD absorbió al fabricante de GPUs ATI, desde entonces está en el aire el proyecto de procesador híbrido de nombre *FUSION*.

[16] Éste kit incluye al otro descendiente de Brook: el lenguaje *Brook+*.

emplearemos BrookGPU, y el paradigma de programación de *«stream processing»*, como marco para el estudio de la implementación más óptima de la FFT en la plataforma de la GPU, el cual es el objeto del capítulo III.

La transformada de complejidad lineal que se obtendrá se implementará, para su aplicación a la extracción de distancias, ya en el capítulo IV, usando multiples GPUs programadas con el lenguaje CUDA, aparecido a finales de 2007.

Capítulo 3

FFT sobre GPU

Para llegar a una implementación optimizada de la FFT sobre GPU hemos de estudiar antes el problema computacional en sí: la FFT; y la máquina —la GPU— y, la abstracción y el lenguaje elegidos para programarla —el paradigma de *stream processing* y cómo éste se concreta en el lenguaje BrookGPU—. Un conocimiento exhaustivo de dichos puntos nos permitirá revisar las contribuciones anteriores de otros autores y proponer una mejora a los mismos.

Este capítulo trata esos puntos en este orden: comenzaremos exponiendo los trabajos previos que han abordado la implementación de la FFT en GPU, profundizaremos en el paradigma de programación que abstrae las GPUs de su forma actual fundiéndola con el resto de arquitecturas basadas en paralelismo a nivel de datos: el *stream processing* [Buck 03a], [Buck 05], antes de pasar al problema computacional en sí y proponer nuestra implementación.

3.1. Trabajos anteriores

El factor limitante del empleo de la tecnología de la GPU al procesamiento de señales se encuentra en el bajo rendimiento obtenido por las implementaciones existentes de la *Transformada Rápida de Fourier*, FFT, sobre ella. Autores como Moreland [Moreland 03], Wokla [Wloka 03] o Viola [Viola 04] no logran mejorar el rendimiento de la libreria estándar sobre CPUs, FFTW [Frigo 05], a la hora de ejecutar este algoritmo, clave en el procesamiento de señales, pese a lo evidenciado en otros campos de aplicación, donde la GPU sí bate ampliamente a la CPU. La raíz del problema se encuentra en la mala adaptación del algoritmo de la FFT al paradigma de *«stream processor»*. Esta «adaptabilidad», como apuntan Fatahalian *et al.* [Fatahalian 04], depende fuertemente de la relación entre comunicación

de datos frente a cómputo sobre los mismos que requiere cada algoritmo en concreto, lo que se ha denominado «intensidad aritmética». El algoritmo de la FFT, especialmente en la variante original de Cooley y Tukey [Cooley 65] elegida por los autores reseñados, presenta una intensidad aritmética baja.

Un trabajo de Krüger y Westermann, de la *Technische Universität München*, para la implementación eficiente de operadores de álgebra lineal sobre la GPU [Krüger 03], logra un rendimiento entre 12 y 15 veces superior al del paquete BLAS [Dongarra 02] sobre la CPU. Este buen resultado permite, al mismo grupo de investigación, implementar la FFT sobre GPU con mejor rendimiento basándose en la posibilidad de formular el cómputo de la FFT como una operación algebráica sobre matrices dispersas [Schiwietz 04]. Las distintas formas de expresar la matriz de la *Transformada Discreta de Fourier*, DFT, como matriz dispersa da lugar a las variantes de la FFT [Nussbaumer 82]. Otra teoría unificadora de las variantes de la FFT, basada en productos tensoriales, se documenta en Van Loan [van Loan 92]. La teoría y formulación introducida en estos textos es especialmente poderosa en tanto que permiten construir motores automáticos de búsqueda del mejor algoritmo para una arquitectura dada, como es el caso de SPIRAL [Püschel 05], y la propia FFTW [Frigo 05].

Otra manera de expresar las variantes de la FFT consiste en descomponer los índices que aparecen en los sumatorios de la DFT y reducir el número de operaciones haciendo uso de las propiedades de las exponenciales complejas discretas en un enfoque *divide y vencerás*. Esta técnica se puede encontrar en [Swarztrauber 87].

Una variante que se ha mostrado especialmente eficaz a la hora de ser implementada sobre la GPU es la variante de Pease [Pease 68], como recogen el artículo de Jansen [Jansen 04] y la implementación de Horn [Horn 05][1], precisamente por cómo esta variante redistribuye las comunicaciones y operaciones sobre los datos, de forma más ventajosa que en la variante original. Por último, haciendo uso de las mejoras tecnológicas de las GPUs [OpenGL ARB 04], Sumanaweera y Liu [Sumanaweera 05] logran duplicar en rendimiento a la FFTW, a pesar de usar la variante de C&T, de demostrado bajo rendimiento sobre esta plataforma.

También es de destacar la implementación que el fabricante ATI ha realizado específicamente optimizada para sus tarjetas en [ATI 03], y discutido por Mitchell *et al.* en [Mitchell 03]. Más recientemente, AMD-ATI tiene previsto incluir como parte de su *AMD Core Math Library for GPUs, AMCL-GPU*, una librería de FFT para su plataforma *StreamComputing*, aunque

[1] Una versión inicial de esta implementación se distribuye como código de ejemplo del lenguaje BrookGPU.

de momento no ha sido liberada. Por su parte el fabricante NVIDIA incluye en CUDA, si bien como código objeto, una librería FFT específica para esa plataforma. También existe una librería desarrollada por la *Universidad de North Carolina* que implementa la FFT 1D [Govindaraju 06].

3.2. *Stream processing* en GPU: *BrookGPU*

Las GPUs han estado íntimamente ligadas a la síntesis gráfica por ordenador, y los métodos y lenguajes propios de ese ámbito, hasta que la evolución experimentada a nivel hardware ha hecho imprescindible la correspondiente evolución en la programabilidad de la plataforma. El codiseño del hardware —con características tanto orientadas a gráficos como no específicas de este ámbito—, a la par que la infraestructura software de soporte —igualmente orientada a gráficos, o por contra, de propósito general—, no se ha dado hasta recientemente con la aparición de CUDA, *Compute Unified Device Architecture* [NVIDIA Corp. 07].

Hasta entonces, para acceder a la potencia de cómputo subyacente era necesario dominar las APIs de programación gráfica como OpenGL [Segal 03] y los lenguajes de programación a nivel de vértice y pixel (*vertex/fragment programming*), tales como Cg [Fernando 03]. Adicionalemente, si se quería extraer el máximo rendimiento, se debía tener un amplio conocimiento de las extensiones propias de los distintos fabricantes. Las especificaciones de estos añadidos, [Kilgard 04], superaban en longitud la propia del estándar [Segal 03].

Antes de la aparición de CUDA, sin embargo, ya habían surgido iniciativas como el metalenguaje BrookGPU [Buck 04] que, sobre la capa de programación específica para gráficos, montaban una abstracción a más alto nivel que permitía programar las GPUs escondiendo al desarrollador las interioridades de la misma.

En este sentido, no solo la programación de la GPU se desliga de la programación gráfica, sino que los desarrolladores pueden centrarse en la elección de los algoritmos más adecuados al nuevo modelo, y que sean, por tanto, perdurables entre las distintas generaciones de GPUs, así como aprovechables en otras plataformas también basadas en paralelismo de datos, dejando las peculiaridades de cada una y el *gap* entre optimización y generalidad bajo la responsabilidad de los desarrolladores de BrookGPU.

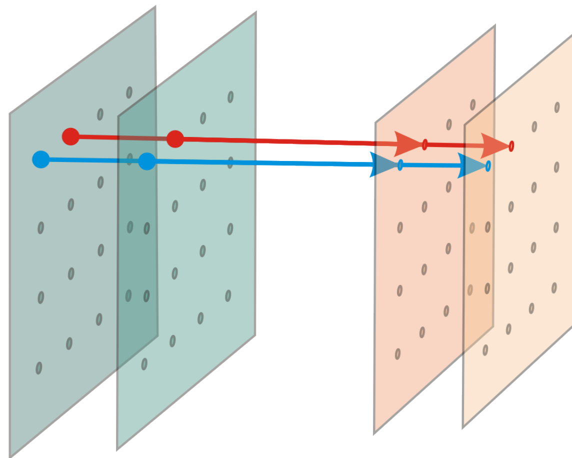


Figura 3.1: Mapeado directo entre *strels* de entrada y salida. Ilustrado con dos *streams* 2D a la entrada, en verde, y otros dos a la salida, en naranja. El mismo programa, *kernel*, se ejecuta para cada *strel*. El cómputo sobre la «posición» marcada por la línea roja es independiente a la coloreada en azul. Cada instancia de ejecución correrá en un *thread* independiente. El uso de flechas quiere recalcar también la direccionalidad del cómputo: no hay *streams* de lectura/escritura. Si son de entrada, se lee de ellos; y si son de salida, se escribe en ellos.

3.2.1. *Streams* y *kernels*

En este paradigma de programación las estructuras de datos se denominarán *streams* y las rutinas que operen sobre las mismas se denominarán *kernels*. Un *kernel* puede tener varios *streams* de entrada, y, desde la aparición de la extensión *Multiple Render Targets, MRT* [OpenGL ARB 04], también puede tener hasta 4 *streams* a la salida.

Los *kernels* generan cada elemento del *stream* de salida operando en los elementos de los *streams* de entrada que «mapean» sobre la posición que está siendo computada. Esto es, cuando se está generando el «elemento de *stream*», *«strel»* a partir de aquí, que ocupa la posición i-ésima del *stream* de salida, el kernel tiene en cuenta exclusivamente los *strels* de entrada posicionados igualmente en i. Este caso de mapeo directo se ilustra en la figura 3.1.

Las entradas de datos a los *kernel*, pueden ser de otros dos tipos: constantes escalares y *gather streams*. Los *gather streams* violan la regla anteriormente enunciada, pudiendo ser consultado su *strel* i-ésimo desde cualquier posición del *stream* de salida. En cualquier caso, su uso no está recomendado puesto que, aunque no son necesariamente más lentos que los *streams* de mapeo directo, estos precisamente refuerzan la coherencia de cache en

los accesos, el cual es un recurso limitado en las GPUs, y determinante del rendimiento que se obtenga.

Existe también un tipo especial de *kernel*, los *kernels* de reducción, los cuales combinan, mediante un operador conmutativo, como puede ser la suma, o el operador mínimo o máximo, varios *strels* a la entrada en uno de salida, cuyo tamaño por tanto se ve reducido. Aunque son una herramienta poderosa conceptualmente, su uso queda igualmente desaconsejado por su pobre rendimiento.

Los *streams* pueden ser 1D, 2D y 3D, y cada *strel* albergar entre 1 y 4 floats. Un tamaño máximo típico para un *stream* 2D es 4096×4096.

Las reglas aplicadas por BrookGPU para encajar los distintos tamaños de los *streams* participantes en un *kernel* son esenciales para entender el mapeo entre *strels* de entrada y salida, y son las siguientes:

- Si el *kernel* tiene varios *streams* de salida, todos ellos han de tener la misma forma (el mismo número de dimensiones y el mismo tamaño en cada dimensión). Y se entiende que esta forma común, es la forma del *kernel*.
- Los *streams* de entrada han de tener igual número de dimensiones que el *kernel*.
- Para cada *stream* de entrada, y para cada dimensión individualmente, se aplican las siguientes reglas para hacer coincidir los tamaños en esa dimensión:
  - si el tamaño del *stream* es mayor que la del *kernel* en un factor entero N, tan solo 1 de cada N *strels* participará en el kernel. A esto se le conoce como *striding* implícito; e.g. $stride_{(N=2)}\{abcd\} \rightarrow \{ac\}$
  - si el tamaño del *kernel* es mayor que la del *stream* de entrada en un factor entero N, cada *strel* se repite N veces. A esto se le conoce como repetición implícita; e.g. $repeat_{(N=2)}\{abcd\} \rightarrow \{aabbccdd\}$

Antes de que estas reglas sean aplicadas, los *streams* que participan en un *kernel* pueden pasar por un operador de *dominio*. Este operador permite la selección de una región del total del *stream*, indicando su comienzo y su final; e.g. $domain_{(start=2,end=3)}\{abcd\} \rightarrow \{cd\}$. El operador de dominio modifica la forma en que el *stream* al que se aplica es mapeado en ese *kernel* en particular sin coste adicional.

Los operadores de dominio pueden verse como la versión explícita con que el programador puede afectar el comportamiento implícito de las reglas

de *stride* y repetición. Haciendo un uso adecuado de ambas, se pueden lograr patrones de mapeo más elaborados que el simple mapeo directo. Por ejemplo, llamar a un *kernel foo* con dos *streams* inX y $outY$, ambos de la forma $N \times M$, con los siguientes operadores de dominio:

$$\begin{aligned} foo\ (\quad & domain_{start=(1,0),end=(N+1,M)}\{inX\}, \\ & domain_{s=(0,0),e=(N/2,M)}\{outY\} \qquad) \end{aligned}$$

primero, afecta a la forma del kernel, que será la del *stream* de salida $outY$: $N/2 \times M$; esto implica que solo la primera mitad vertical de $outY$ se computará en el kernel *foo*; el operador de dominio sobre inX realiza un desplazamiento vertical de un elemento, pero no modifica su forma, y por lo tanto, se verá afectado por un *stride* implícito que descartará la mitad de las filas, para así acomodarse a la forma del *kernel*; el efecto de este operador es que sean las filas impares de inX las que participarán en el *kernel*, en lugar de las pares, que lo hubieran hecho de no mediar el uso del operador de dominio.

3.2.2. Ejemplo de uso de *kernels* y *streams* en *BrookGPU*

Uno de los pasos en el algoritmo de la transformada de Fourier discreta es el intercambio de aquellos datos cuyas posiciones son la una inversa bit a bit de la otra. A esta operación se la conoce como *index bit-reversal*. La usaremos para ejemplificar el uso de *kernels* y *streams* en BrookGPU.

En el ejemplo, se asume que un *strel* alberga dos datos complejos que en la secuencia original son adyacentes *lsb*, *least significant bit*, esto es, sus posiciones solo difieren en el bit menos significativo.

¿Cómo realizaremos la inversión de bits en los índices de una secuencia 1D? Primero, se parte por la mitad la secuencia de tamaño N, dando lugar a dos secuencias de tamaño $N/2$. Considerados aisladamente los índices en una y otra mitad son iguales tras la partición, cuando antes diferían en el bit más significativo, *«Most Significant Bit», MSB*. En segundo lugar, se intercambian los datos cuyos índices estén invertidos en binario, usando para ello el mismo patrón de inversión en ambas mitades. Por último, la secuencia final se logra fusionando ambas partes cogiendo alternadamente un dato desde cada una:

$$0123\ 4567 \ \rightarrow \overset{0}{0}123|\ \overset{4}{0}123 \ \rightarrow \overset{0}{0}213|\ \overset{4}{0}213 \ \rightarrow \ 04\ 26\ 15\ 37$$

En términos de *streams* y *kernels* la inversión se realiza a dos niveles. El intercambio de posiciones debidas a la inversión $MSB \leftrightarrow lsb$ se hace separa-

damente, usando operaciones de *stream*, del intercambio en el resto de bits, que se basará en un patrón prefijado de *gather*.

En los siguientes trozos de código escrito en BrookGPU, al *stream* 2D, `X`, compuesto por elementos `float2` se le aplica un *index bit-reversal* bidimensional (aunque la idea a aplicar se explicara unidimensionalmente, la extrapolación es directa) y a la vez se le empaqueta en un *stream* `Xr` de elementos `float4` con la mitad de elementos en horizontal. El *stream* `br` contiene el patrón de *bit-reversal*. Nótese el mecanismo para cargar datos a los *streams*, y la manera en que las sentencias propias de BrookGPU se insertan en un código C:

```
float2 X<N,M>, br<N, M/2>;                   // streams
float4 Xr<N, M/2>;
float2 data_br[N][M/2], data_X[N][M];        // arrays

for (i=0; i < N; i++)
  for (j=0; j < M/2; j++) {
    data_br[i][j].x = bitReverse(j);
    data_br[i][j].y = bitReverse(i);
  }
dataRead(br, data_br);
dataRead(X, data_X);
```

La declaración del *kernel* que realiza el *index bit-reversal* y el empaquetado de las dos secuencias recibe 3 argumentos: las dos subsecuencias, instanciadas como *gather streams*, y las posiciones precomputadas desde las cuales se deben traer esas entradas, instanciadas como un *stream* normal (no *gather*).

El código del kernel en sí tan solo remite los números complejos traidos desde la primera y segunda mitad del *stream* original al *slot* apropiado (`.xy` o `.zw`) del *strel* de salida:

```
kernel void reverse(float2 X_0_M2[][],
                    float2 X_M2_M[][],
                    float2 br<>,
                    out float4 result<>) {
  result.xy = X_0_M2[br];
  result.zw = X_M2_M[br];
}
```

Los operadores de dominio, aplicado en la llamada al *kernel*, dividen horizontalmente el *stream* `X`:

```
reverse(X.domain(int2(0,0), int2(M/2,N)),
        X.domain(int2(M/2,0), int2(M,N)),
```

```
br, Xr);
```

La división se ha logrado haciendo un uso adecuado del operador de dominio, mientras que la fusión se realizó mediante un *kernel* que redireccionó los datos.

3.3. Transformada rápida de Fourier

3.3.1. De la DFT a la FFT

La transformada discreta de Fourier, *F(k)*, de una secuencia 1D de números complejos *f(n)* de tamaño N, se calcula mediante

$$F(k) = \sum_{n=0}^{N-1} f(n) \cdot W_N^{kn}, \qquad 0 \leq k < N \tag{3.1}$$
$$W_N^{nk} = e^{-i2\pi \frac{kn}{N}}$$

Si $N = 2^r$ entonces los índices n y k se pueden expresar en binario usando la función de índice I:

$$n = I(n_{r-1}, n_{r-2}, \ldots, n_1, n_0) = 2^{r-1}n_{r-1} + 2^{r-2}n_{r-2} + \cdots + 2n_1 + n_0; \tag{3.2a}$$

$$k = I(k_{r-1}, k_{r-2}, \ldots, k_1, k_0) = 2^{r-1}k_{r-1} + 2^{r-2}k_{r-2} + \cdots + 2k_1 + k_0 \tag{3.2b}$$

Al sustituir estos índices en la ecuación (3.1) resulta

$$F(k_{r-1}, \ldots, k_0) = \sum_{n_{r-1}=0}^{1} \cdots \sum_{n_0=0}^{1} f(n_{r-1}, \ldots, n_0) \cdot W_{2^r}^{(2^{r-1}n_{r-1}+\cdots+n_0)(2^{r-1}k_{r-1}+\cdots+k_0)} = \sum_{n_0=0}^{1} \cdots \sum_{n_{r-1}=0}^{1} f(n_{r-1}, \ldots, n_0) \cdot W_{2^r}^{\sum_{i=0}^{r-1}(2^i n_i \cdot \sum_{j=0}^{r-1} 2^j k_j)}$$

Las exponenciales complejas discretas verifican que $W_{2^r}^{2^r z} = 1$, para todos los enteros $z \in \mathbb{Z}$, y, por tanto, todas las combinaciones $2^i n_i \cdot 2^j k_j$, con $i+j \geq r$

se pueden cancelar, obteniéndose el número mínimo de operaciones. Hemos llegado ya a la expresión de la transformada discreta rápida de Fourier:

$$F(k_{r-1},\dots,k_0) = \sum_{n_0=0}^{1}\dots\sum_{n_{r-1}=0}^{1} f(n_{r-1},\dots,n_0)\cdot W_{2^r}^{\sum_{i=0}^{r-1}\left(2^i n_i\cdot\sum_{j=0}^{r-1-i}2^j k_j\right)}$$
$$= \sum_{n_0=0}^{1}\dots\sum_{n_{r-1}=0}^{1} f(n_{r-1},\dots,n_0)\cdot W_2^{n_{r-1}k_0}\cdot\;\dots\;\cdot W_{2^{r-1}}^{n_0\cdot\left(\sum_{j=0}^{r-1}2^j k_j\right)}$$

Cabe considerar cómo llevar a cabo estos cómputos de manera eficiente. Para verlo recurriremos a una secuencia de ejemplo de tamaño $N=8,\ r=3$. Se comienza cancelando los términos innecesarios:

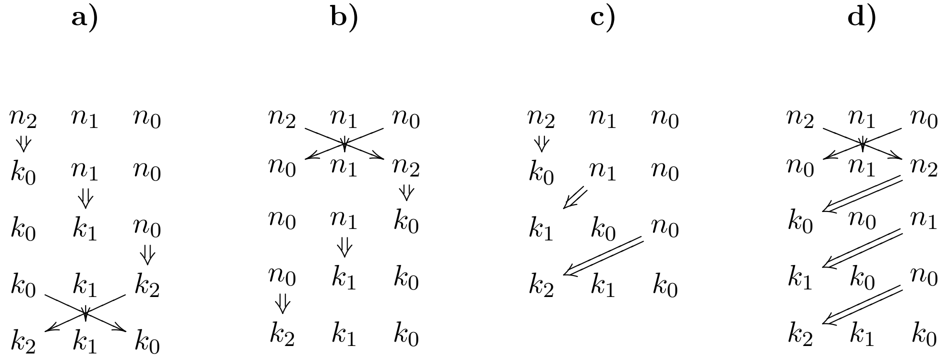
$$\sum_{n_0=0}^{1}\sum_{n_1=0}^{1}\sum_{n_2=0}^{1} f(n_2,n_1,n_0)\cdot W_8^{(2^2k_2+2^1k_1+2^0k_0)(4n_2+2n_1+n_0)},$$
$$\sum_{n_0=0}^{1}\sum_{n_1=0}^{1}\sum_{n_2=0}^{1} f(n_2,n_1,n_0)\cdot W_8^{4k_2(\cancel{4n_2}+\cancel{2n_1}+n_0)+2k_1(\cancel{4n_2}+2n_1+n_0)+k_0(4n_2+2n_1+n_0)},$$
$$\sum_{n_0=0}^{1}\sum_{n_1=0}^{1}\sum_{n_2=0}^{1} f(n_2,n_1,n_0)\;\cdot\;W_8^{k_0(4n_2+2n_1+n_0)}\;\cdot\;W_4^{k_1(2n_1+n_0)}\;\cdot\;W_2^{k_2n_0}$$

Ahora se tiene, que el sumatorio interno, con índice n_2, se puede colapsar para dar lugar a una nueva secuencia de datos, $F^{1)}$, que queda en términos de los restantes índices del dominio de partida, n_1 y n_0, y del nuevo índice en el dominio transformado, k_0:

$$\sum_{n_0=0}^{1}\sum_{n_1=0}^{1}\overbrace{\left\{\sum_{n_2=0}^{1} f(n_2,n_1,n_0)\cdot W_8^{k_0(4n_2+2n_1+n_0)}\right\}}^{F^{1)}(k_0,n_1,n_0)}\cdot W_4^{k_1(2n_1+n_0)}\cdot W_2^{k_2n_0}$$

Para llegar a $F(k)$ esta operación se ha de repetir r veces, 3 en el ejemplo. En el caso general, en la etapa l se realiza la «sustitución» de un índice binario del dominio de partida por uno del dominio transformado: $n_{r-l}\Longrightarrow k_{l-1}$. Si continuamos con nuestro ejemplo:

$$\sum_{n_0=0}^{1}\overbrace{\left\{\sum_{n_1=0}^{1} F^{1)}(k_0,n_1,n_0)\cdot W_4^{k_1(2n_1+n_0)}\right\}}^{F^{2)}(k_0,k_1,n_0)}\cdot W_2^{k_2n_0},$$



Cuadro 3.1: Variantes de la FFT. De izquierda a derecha: **a)** Variante de Cooley & Tukey radix-2 con decimación en frecuencia, y **b)** con decimación en tiempo; **c)** de Stockham; y **d)** de Pease

$$\underbrace{\sum_{n_0=0}^{1} F^{2)}(k_0, k_1, n_0) \cdot W_2^{k_2 n_0}}_{F^{3)}(k_0,k_1,k_2)}$$

En definitiva se ha reformulado el problema para tratarlo como una transformada multidimensional, con dimensiones binarias, que luego se han procedido a transformar independientemente, siguiendo un orden determinado, en un proceso de varias etapas.

La secuencia finalmente obtenida $F^{3)}(k_0, k_1, k_2)$ no es aún exactamente la transformada discreta de la secuencia a la entrada. Los índices binarios una vez transformados han ocupado, en la lista de parámetros de la nueva secuencia a la que daban lugar, la misma posición que el índice al que sustituían. Por lo que han terminado invertidos respecto a lo deseado, dado que se transforma $n_{r-l} \Longrightarrow k_{l-1}$, requiriendo, como paso final, una recolocación:

$$\begin{aligned} F(k) &= F(k_2, k_1, k_0) = \\ &= IndexBitReversal(F^{3)}(k_0, k_1, k_2)) \end{aligned}$$

El algoritmo descrito corresponde al propuesto por Cooley y Tukey, C&T, con descomposición en índices de base-2 (*radix-2*). A esta versión del algoritmo, con el *bit-reversal* aplicado al final, sobre los índices del dominio transformado, se le denomina *«decimation-in-frequency», DIF*. En contraposición a haberlo realizado inicialmente: *«decimation-in-time», DIT*.

3.3.2. Variantes de la FFT

El orden en que se ha de realizar la transformación de índices viene dado por las dependencias en el cómputo, y es fijo: $n_{r-1} \Longrightarrow k_0$, ..., $n_0 \Longrightarrow k_{r-1}$. Lo que si se puede alterar, sin alterar el resultado final, es la «posición» que ocupa en la secuencia $F^{l-1)}$ el índice binario que «desaparece», n_{r-l}, y por dónde «aparece» en $F^{l)}$ el índice k_{l-1}. Estas combinaciones dan lugar a las variantes de la FFT.

En el cuadro 3.1 se muestran de una manera intuitiva las distintas variantes para una secuencia de tamaño 8. En **a)** y **b)** se esquematiza la variante de Cooley & Tukey [Cooley 65], y cómo realiza sustituciones de índice *in-place*, por lo que requiere un *index bit-reversal* que puede ser realizado o bien al comienzo, DIT, o como paso final, DIF. En **c)** se muestra la variante de Stockham [Stockham 66]. La sustitución de índices es *out-of-place*, pero a cambio no necesita una etapa diferenciada de intercambio de índices. De hecho, este paso se ha entremezclado con el cómputo, realizándose, al igual que aquel, progresivamente. **d)** Variante de Pease [Pease 68]. Los movimientos de datos no varían con la etapa: los índices de partida desaparecen de la posición menos significativa, y el correspondiente índice transformado, aparece en la más significativa. Existe también la variante de Pease con decimación en frecuencia. En ese caso, n_{r-l} ocuparía siempre la posición más significativa de su secuencia, y k_{l-1} la menos significativa.

Las sutiles implicaciones que se desprenden de elegir una u otra variante se entienden mejor si extendemos el índice del sumatorio de una etapa cualquiera l:

$$F^{l)}(k_{l-1},\ \overbrace{k_{l-2},\ldots,k_0}^{\kappa:\, l-1\, bits},\ \overbrace{n_0,\ldots,n_{r-l-1}}^{\nu:\, r-l\, bits}) = \sum_{n_{r-l}=0}^{1} F^{l-1)}(\kappa,\ \nu,\ n_{r-l}) \cdot W_{2^{r-l+1}}^{k_{l-1}\left(\sum_{i=0}^{r-l} 2^i n_i\right)} = \left[F^{l-1)}(\kappa,\ \nu,\ 0) + F^{l-1)}(\kappa,\ \nu,\ 1) \cdot \underbrace{W_2^{k_{l-1}}}_{1\text{ ó }-1}\right] \cdot \underbrace{W_{2^{r-l+1}}^{k_{l-1}\left(\sum_{i=0}^{r-l-1} 2^i n_i\right)}}_{1\text{ ó } twiddle_\nu} \quad (3.3)$$

Resulta que para computar un elemento de $F^{l)}$ se requieren dos elementos de $F^{l-1)}$. Es más, dos elementos de $F^{l)}$, correspondientes a $k_{l-1} = \{0,\ 1\}$, pueden ser computados con los dos datos de $F^{l-1)}$, $n_{r-l} = \{0,\ 1\}$, y el adecuado factor de *twiddle*:

$$F^{l)}(0,\kappa,\nu) = F^{l-1)}(\kappa,\nu,0) + F^{l-1)}(\kappa,\nu,1) \quad (3.4a)$$

$$F^{l)}(1,\kappa,\nu) = \left[F^{l-1)}(\kappa,\nu,0) - F^{l-1)}(\kappa,\nu,1)\right] \cdot twiddle_\nu \quad (3.4b)$$

Las ecuaciones anteriores, propias de la variante de Pease, se convierten, para la de C&T con DIF, en:

$$F^{l)}(\kappa,0,\nu) = F^{l-1)}(\kappa,0,\nu) + F^{l-1)}(\kappa,1,\nu) \tag{3.5a}$$

$$F^{l)}(\kappa,1,\nu) = \left[F^{l-1)}(\kappa,0,\nu) - F^{l-1)}(\kappa,1,\nu)\right] \cdot twiddle_{\nu} \tag{3.5b}$$

Los dos elementos de cada secuencia $F^{l)}$ y $F^{l-1)}$ que participan en estas ecuaciones se denominan *nodos duales*, y se conoce como *mariposa* a la operación que las relaciona. La distancia en memoria, el *stride*, entre los dos elementos de partida de una mariposa viene dado por la significancia del índice binario que desaparece. Mientras que el *stride* en la secuencia resultado viene dado por la significancia con la que aparece el nuevo índice transformado.

La primera sustitución en el esquema C&T DIF, en que ambos índices aparecen y desaparecen en la posición más significativa implicará traer pares de datos distanciados $N/2$ (el peso en la función I del índice que está siendo transformado), combinarlos usando la exponencial compleja discreta adecuada, y dejar el resultado en las mismas posiciones que ocupaban los datos inicialmente, separados $N/2$ entre sí.

En cambio, una sustitución como las que se dan en la variante de Pease, donde los índices desaparecen de la posición menos significativa y aparecen en la más significativa, implica reunir datos con un *stride* unitario, y depositarlos con un *stride* máximo. La variante de Stockham distribuye la decimación entre todas las etapas.

Tan solo en la variante de Pease los *strides* antes y después del cómputo son independientes de la etapa en la cuál nos encontramos. Nótese al respecto, en las ecuaciones (3.4) y (3.5), que aunque el número de bits en las subcadenas de índices κ, cuánto se ha transformado ya, y ν, cuánto resta por transformar, varían siempre con la etapa, el hecho de que en la variante de Pease estas subcadenas aparecen juntas la una a la otra, en la forma κ, ν, n_{r-l}, o k_{l-1}, κ, ν, por lo que su longitud total permanece constante.

La condición *out-of-place* de las variantes de Pease y Stockham pueden ser parcialmente evitadas si el cómputo se realiza *in-place* y posteriormente se recombina adecuadamente:

$$\begin{array}{ccccccccc}
k_{l-2} & \cdots & k_0 & n_0 & \cdots & n_{r-l-1} & n_{r-l} & & \\
 & & & & & & \Downarrow & & \\
k_{l-2} & \cdots & k_0 & n_0 & \cdots & n_{r-l-1} & k_{l-1} & & \\
k_{l-1} & k_{l-2} & \cdots & k_0 & n_0 & \cdots & n_{r-l-1} & &
\end{array} \tag{3.6}$$

Esta manera de explicar las variantes de la FFT difiere de lo expuesto en [Nussbaumer 82] o [van Loan 92], si bien todas ellas son notaciones equipotentes. La aquí descrita nos permitirá extender a bidimensional el algoritmo

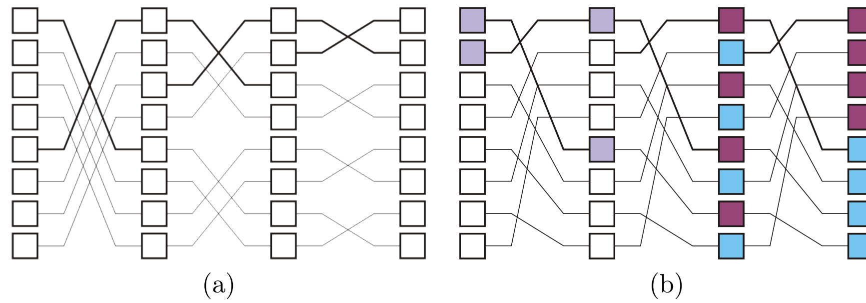


Figura 3.2: Mariposas de las variantes **a)** de Cooley&Tukey, y **b)** de Pease.

propuesto de manera más sencilla que de haber empleado aquellas otras, o la descripción gráfica habitual de flujo de datos en mariposa, que se muestra en la figura 3.2. En dicha figura se aprecia como la distancia entre nodos duales en la variante de C&T cambia de etapa a etapa, pero no varía entre entrada y salida: los cómputos pueden ser realizados *in-place*. En la variante de Pease, los cómputos son *out-of-place* pero las mariposas tienen un patrón fijo para todas las etapas: véase cómo los nodos en violeta siempre se combinan entre sí. Las combinaciones a la entrada son entre nodos cuyas posiciones difieren en el bit menos significativo, y por contra, a la salida lo son entre vecinos alejados en el peso del bit más significativo: $N/2$. Aunque el patrón parezca más irregular que en C&T, los movimientos de datos se pueden explicar a nivel global de manera más sencilla, como se explicará más adelante, usando el coloreado en morado y celeste de los nodos de la última etapa.

3.3.3. Elección de la variante de Pease

La optimalidad de la implementación que se haga de la FFT sobre el paradigma de programación de *stream* dependerá de la variante de la FFT que se escoja. Esta elección ha de tener en cuenta los puntos fuertes de las GPUs mientras evita aquellos aspectos que hacen caer su rendimiento.

Esto hará que pongamos el esfuerzo en mover y operar los datos aprovechando la configuración 4-way SIMD de las GPUs, esto es operando en streams del tipo `float4`; empleando mapeo directo, o en su caso operadores de dominio y reacomodación implícita, en lugar de *gather streams*; evitando los *kernels* de reducción; y llevando a cabo en la CPU aquellos cómputos imposibles o demasiado costosos para la GPU.

Es conveniente analizar el coste computacional de las distintas variantes de la FFT del cuadro 3.1 en términos de *streams*. El *stride* entre datos en las variantes de C&T y Stockham dependen de la etapa en curso. Trasladar estos *strides* a mapeo de *streams*, implica necesariamente el uso de *gather streams*

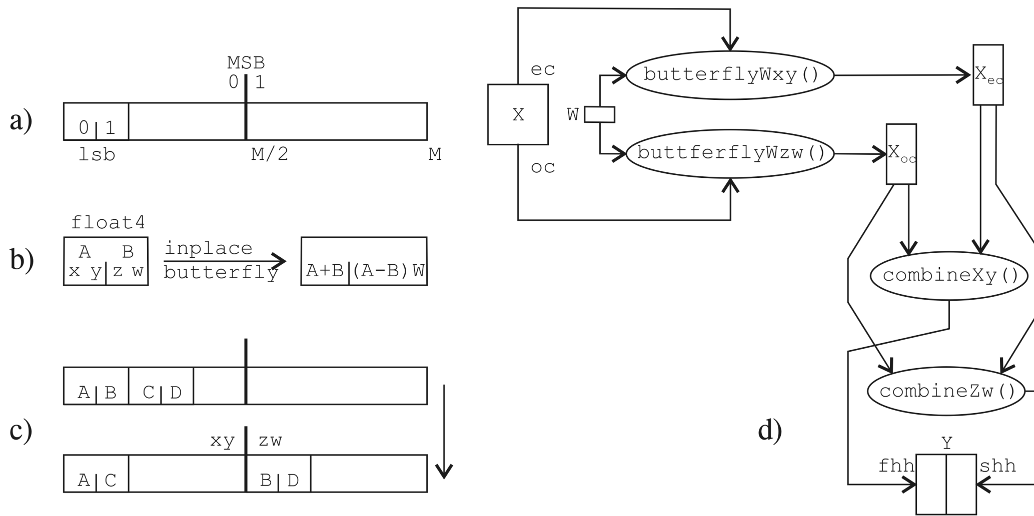


Figura 3.3: FFT de multiples secuencias 1D almacenadas como filas de un *stream* 2D.

que indiquen para cada etapa y posición del *stream*, dónde ir a buscar su nodo dual. En contraste con este enfoque, la variante de Pease tiene una propiedad notable: el *stride* de entrada de su mariposa es unitario e invariante con la etapa. Esta propiedad permite realizar la operación de mariposa *radix-2* dentro de un *strel*, puesto que dos números complejos, con sus correspondientes parte real e imaginaria, se pueden almacenar en un *strel* `float4`.

De tal manera que para computar una mariposa sobre los dos elementos de un *strel*, tan solo es necesario consultar la exponencial compleja adecuada, pero ningún otro elemento del *stream* de datos. Así, si los resultados se vuelven a almacenar *in-place*, se puede aplicar un mapeo directo entre la secuencia, *stream*, de entrada y una secuencia de salida, que aún no sería definitiva. Pero este paso se habría realizado haciendo un aprovechamiento óptimo del ancho de banda de memoria de la GPU. El inconveniente es que restaría realizar la recombinación de los datos, como se adelantó en el esquema (3.6), para llevar los datos, que tras el cómputo *in-place* permanecen *lsb*-adyacentes, a un *stride MSB*. La recombinación se puede llevar a cabo de manera similar a como se realiza el *index bit-reversal*, que se puso como ejemplo en la sección 3.2.2.

En la subfigura *a)* de la figura 3.3 se representa la distancia, sobre una fila del *stream*, de los nodos cuyas posiciones difieren en el bit menos significativo o en el más significativo, *least/Most significant bits* (*lsb, MSB*).

En *b)* se aprecia como dividiremos lógicamente un *strel* `float4` en dos *slots*: `xy` y `zw`. Para realizar una operación *in-place* de una mariposa se re-

quieren los nodos duales, `A` y `B`, que comparten un mismo *strel*, y el correspondiente factor *twiddle* `W`. Las operaciones aritméticas se realizarán con instrucciones *4-way SIMD*.

En *c)* tenemos cómo se redistribuirán los elementos de una fila del *stream* de datos: los vecinos *lsb* `A`, `B` terminan estando a distancia *MSB* ubicados en el primer *slot* de sus correspondientes *strels*. `C` y `D`, que comenzaron ocupando una columna impar del *stream* se reubican a los segundos *slots* de sus *strels*.

Por último en *d)* se esquematiza el funcionamiento de la variante de Pease usando una representación gráfica de los *streams* y *kernels* con los que se realiza el cómputo. Es conveniente volver sobre la figura 3.2, donde se aprecia más claramente que en la subfigura *c)* las implicaciones de esta variante: sólo con ella se puede llegar a un mapeo *no-gather* en todas las etapas, recurriendo exclusivamente a operadores de dominio a nivel de todo el *stream*. Las etiquetas `ec` y `oc` en las flechas de entrada a los kernels representan los operadores de dominio que seleccionan las columnas pares (***e**ven **c**olumns*) o impares (***o**dd **c**olumns*). De manera análoga `fhh` y `shh` representan a la selección de la primera o segunda mitad horizontal (***f**irst/**s**econd **h**orizontal **h**alves*).

El punto más complicado en la implementación de la FFT sobre la GPU es trasladar el flujo de datos de las mariposas al modelo de programación de *stream*. En cambio implementar las mariposas, ecuaciones (3.4), en un *kernel* que haga uso del juego de instrucciones 4-way SIMD de las GPUs es trivial.

Tampoco resulta sencillo computar las exponenciales complejas $W(N, n, k, l)$, puesto que no solo se requiere operar con funciones trigonométricas, sino también realizar operaciones a nivel de bit que dependen de la posición y de la etapa, y para las que no estaban preparadas las primeras GPUs. Se obtiene un mejor rendimiento si estos factores de *twiddle* se precomputan en la CPU, se transfieren luego a la GPU, y se consultan por mapeo directo en el *kernel* de mariposa.

Por lo tanto, la variante de Pease tiene varias ventajas a la hora de ser implementada sobre una GPU, y en general sobre cualquier *stream processor*, respecto a las restantes variantes de la FFT. Y además, sus desventajas pueden ser parcialmente aliviadas usando operadores de dominio junto con *kernels* redireccionadores, sin cómputo.

3.4. Caso unidimensional, múltiples secuencias de tamaño regular

Implementar la FFT para secuencias pequeñas (por debajo de 1×4096) sobre la GPU no mejora el rendimiento del que son capaces las CPUs por el uso que éstas hacen de las memorias caché. En cambio el uso de la GPU es aconsejable si se aumenta el tamaño del problema.

Las técnicas anteriormente esbozadas (*index-bit-reversal*, mariposas *in-place*, recombinación de la salida) computan la FFT 1D cuando se aplican sobre un *stream* de tamaño $1 \times M$. Sin realizar cambio alguno en el algoritmo, de ser aplicadas a un *stream* de tamaño $N \times M$, donde cada fila del *stream* recoge una secuencia 1D de tamaño $1 \times M$, resuelve N FFTs 1D en paralelo.

El diagrama de la subfigura 3.3.*d)* muestra la relación entre *streams* y *kernels* que ejecutan una de las $log_2(M)$ etapas requeridas. El *stream* `X` tiene tamaño $N \times M/2$, después del *index bit-reversal* y el empaquetado a `float4` de la secuencia compleja de entrada.

Cuando `X` se divide horizontalmente en $\texttt{X}_{\texttt{ec}}$ y $\texttt{X}_{\texttt{oc}}$, los *streams* resultantes presentan un tamaño $N \times M/4$.

Se dispone de dos *kernels* de mariposa que difieren exclusivamente en el *slot* del *stream* `W` que consultan: `.xy` para las columnas pares y `.zw` para las impares. De este modo el *stream* `W` será de tamaño $1 \times M/4$.

Después de computar $\texttt{X}_{\texttt{ec}}$ y $\texttt{X}_{\texttt{oc}}$ se les recombina según se ilustra en la subfigura 3.3.*c)*. Cada *strel* en la primera mitad horizontal de `Y` obtiene su primer *slot* desde el primer *slot* en $\texttt{X}_{\texttt{ec}}$, y el segundo *slot* lo obtiene del primer *slot* en $\texttt{X}_{\texttt{oc}}$. Esto se realiza en el *kernel* `combineXY()`. El *kernel* `combineZW()` combina los segundos *slots* de $\texttt{X}_{\texttt{ec}}$ y $\texttt{X}_{\texttt{oc}}$ en la segunda mitad horizontal de `Y`.

El *stream* `W` cambia con la etapa. Sin embargo, todos los posibles valores a consultar han de estar disponibles en la primera etapa, mientras que cada etapa posterior requiere solo la mitad de la etapa que la precede. Por lo que se pueden computar inicialmente en la CPU, moverse a memoria de la GPU en el momento inicial y luego realizar en GPU la poda de valores de las posteriores etapas. La diversidad decreciente en valores de `W` permite que las dos últimas etapas se realizen por *kernels* simplificados de mariposa que no consultan `W`.

3.5. Caso bidimensional, método *row-column*

Si somos capaces de realizar N FFTs 1D de tamaño M en paralelo, estamos a solo un paso de poder realizar transformadas 2D de tamaño $N \times M$, aplicando la propiedad de separabilidad:

$$F(k_y, k_x) = \sum_{n_y=0}^{N-1} \sum_{n_x=0}^{M-1} f(n_y, n_x)\, W_M^{k_x n_x}\, W_N^{k_y n_y}\,, \quad 0 \le k_y < N,\ 0 \le k_x < M$$
$$= \sum_{n_y=0}^{N-1} \underbrace{\left\{ \sum_{n_x=0}^{M-1} f(n_y, n_x) W_M^{k_x n_x} \right\}}_{F_x(n_y, k_x)} W_N^{k_y n_y} \;=\; \sum_{n_y=0}^{N-1} F_x(n_y, k_x)\, W_N^{k_y n_y} \quad (3.7)$$

La FFT 2D de una secuencia de tamaño $N \times M$ se puede obtener aplicando consecutivamente N transformadas 1D a lo largo de las filas y M transformadas 1D en las columnas.

Si queremos reutilizar exactamente el mismo algoritmo de antes, se necesitaran adicionalmente dos transposiciones de los datos. La primera se aplica tras computar las N FFTs 1D de tamaño $1 \times M$ a lo largo de las filas. La transposición reconfigura la secuencia de datos como un *stream* de tamaño $M \times N$ con los datos transformados ocupando lo que ahora son las columnas. Entonces se repite el múltiple cómputo de las FFTs 1D a lo largo de filas, simplemente cambiando el tamaño de las transformadas, que será ahora $1 \times N$. Esto implica realizar pequeños cambios en el *stream* `W`, que ha de tener tamaño $1 \times max(N, M)$. Por último, otra transposición recupera la forma original de los datos.

El algoritmo se podría reformular para que se ejecutara directamente a lo largo de las columnas, eliminando la transposición intermedia, pero como la adyacencia *lsb* caería fuera de los límites de un *strel* el rendimiento sería inferior.

Luego la única pieza nueva para armar el algoritmo 2D es la operación de transposición. Dado que hemos optado por confinar dos números complejos que son *lsb*-adyacentes horizontalmente en un *strel* `float4`, cuando realicemos la transposición vamos a necesitar traer dos *strels* (que contendrán 4 números complejos), pero de los cuales solo aprovecharemos la mitad (los 2 números complejos que albergará el *strel* transpuesto). Esta operación necesita obligatoriamente emplear *gather streams*: algo así como $X^T = X[indexOf(X^T).yx]$, enrevesado por la asimetría del empaquetado a `float4` en horizontal. Con todo, ese recurso será un mal menor, frente a lo que representaría parar los cómputos en la GPU, comunicar los datos a la CPU, transponer y de nuevo mandar los datos a GPU.

Esta forma de operar para obtener un método 2D a partir de una implementación 1D, conocido como método *row-column*, es por el que optan unánimemente los otros autores que implementan FFTs 2D sobre GPU (y no son todos, puesto que muchos solo estudian el problema 1D). Esta tendencia

tiene su razón de ser en las CPUs, pero se demuestran ineficaces sobre la GPU, por lo que posteriormente daremos un método propiamente bidimensional.

3.6. Caso unidimensional, secuencias grandes alojadas en 2D

En las CPUs es habitual albergar en una memoria 1D arrays que se redireccionan 2D. En la GPU, orientada a texturas 2D, es más usual el mecanismo contrario: albergar secuencias que son tratadas lógicamente como 1D en un espacio de direcciones 2D. Si este espacio 2D de direcciones es de tamaño $N \times M$ (N filas de longitud M), un dato que ocupe en la disposición lógica 1D la posición i, en la disposición «física» 2D se ubicará en `[i / M][i % M]`, donde `/` es el operador división entera, y `%` el operador resto de la división entera, o módulo. O, inversamente, el dato físicamente en `[i][j]`, es el dato lógico `[i*M+j]`.

Recurriendo a esta forma de almacenamiento, una secuencia 1D de gran longitud puede sortear el límite de tamaño que impone la GPU a un *stream* (máximo 4096 por dimensión). Por ejemplo 262k elementos encajarían en un *stream* 2D de tamaño 512×512.

Bajo estas condiciones, la variante de Pease sigue siendo aplicable, si bien, la nueva disposición de los datos ha de ser tenida en cuenta al precomputar `W`, y en la etapa de recombinación.

La subfigura 3.4.*a)* muestra las implicaciones de tal disposición en las adyacencias *lsb* y *MSB*, mientras que 3.4.*b)* contiene las operaciones de *stream* que realizan la recombinación adecuada. En este diagrama, `er` representa la selección de filas pares (***e****ven* ***r****ows*) y `or` a las impares (***o****dd* ***r****ows*). Las etiquetas con un número seguido de una `q` seleccionan un cuadrante particular del *stream* de escritura. La división en horizontal y los *kernels* de mariposa que dan lugar a los *streams* $\mathtt{X_{ec}}$ y $\mathtt{X_{oc}}$ no se incluyen en la figura, por ser exactamente idénticos a los que aparecen en la subfigura 3.3.*d)*.

Aunque, como en el caso 1D, los valores de los exponentes en `W` disminuyen con el avance de las etapas, realizar en GPU la selección de cuáles sobreviven, dada la nueva disposición 2D, consumiría más tiempo que suministrar desde la CPU *streams* precomputados específicamente para cada etapa, `W[l]`.

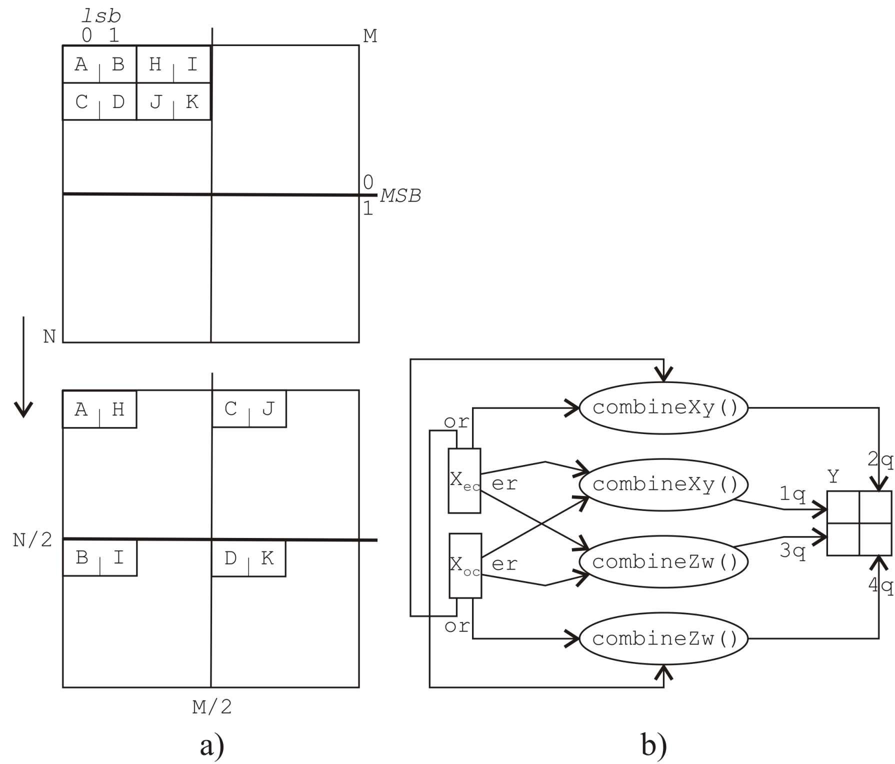


Figura 3.4: Recombinación en la FFT de una secuencia 1D dispuesta en un *stream* 2D. **a)** La frontera *MSB* en este caso se encuentra dividiendo verticalmente el *stream*. Los elementos ubicados en posiciones con $lsb = 0$, tales como `A`, `C`, `H` y `J`, terminarán en la primera mitad vertical del *stream*. Los elementos inicialmente ubicados en columnas pares, tales como `A`, `B`, `C` y `D` se ubicarán en la primera mitad horizontal del *stream* resultante. **b)** *Kernels* y *streams* implicados en el paso de recombinación. Los resultados de los *kernels* de mariposa, los *streams* $\mathtt{X}_{\mathtt{ec}}$ y $\mathtt{X}_{\mathtt{oc}}$, se recombinan en las cuatro subdivisiones del *stream* de salida que marcan las combinaciones *lsb*=0/1 y filas pares/impares. La fase de cómputo es idéntica a la mostrada en la primera parte de la subfigura 3.3.*d)*.

3.7. Caso propiamente bidimensional

Consideremos el problema de realizar la DFT sobre una secuencia 2D $f(n_y, n_x)$, pero ahora sobre ambas dimensiones simultáneamente.

$$F(k_y, k_x) = \sum_{n_y=0}^{N-1} \sum_{n_x=0}^{M-1} f(n_y, n_x) \cdot W_N^{k_y n_y} \cdot W_M^{k_x n_x}, \qquad 0 \le k_y < N, \qquad 0 \le k_x < M \tag{3.8}$$

La secuencia será en general de tamaño N filas $\times$ M columnas, pero empecemos considerando $M = N$ por simplicidad. Como antes, consideramos $N = 2^r$ de tal manera que los índices se pueden descomponer en binario.

La idea que aplicaremos es extender el esquema Pease *radix-2*, que plasma el esquema (3.6) para el caso 1D, para considerar la transformación simultánea de índices en ambas dimensiones:

$$\begin{array}{cccccc} k_{y_{l-2}} & \cdots & n_{y_{r-l}}, & k_{x_{l-2}} & \cdots & n_{x_{r-l}} \\ k_{y_{l-1}} & \swarrow\cdots & n_{y_{r-l-1}}, & k_{x_{l-1}} & \swarrow\cdots & n_{x_{r-l-1}} \end{array} \tag{3.9}$$

La transformación parcial entre etapas del esquema Pease *radix-2–radix-2* queda como sigue:

$$F^{l)}(k_{y_{l-1}}, \overbrace{k_{y_{l-2}}, .., k_{y_0}}^{\kappa_y:\, l-1\ bits}, \overbrace{n_{y_0}, .., n_{y_{r-l-1}}}^{\nu_y:\, r-l\ bits}, k_{x_{l-1}}, \underbrace{k_{x_{l-2}}, .., k_{x_0}}_{\kappa_x}, \underbrace{n_{x_0}, .., n_{x_{r-l-1}}}_{\nu_x}) =$$

$$\sum_{n_{y_{r-l}}}^{0,1} \sum_{n_{x_{r-l}}}^{0,1} F^{l-1)}(\kappa_y,\ \nu_y,\ n_{y_{r-l}}\ ,\ \ \kappa_x,\ \nu_x,\ n_{x_{r-l}}) \cdot W_{2^l}^{\left[n_{y_{r-l}}\left(\sum_{i=0}^{l-1} 2^i k_{y_i}\right) + n_{x_{r-l}}\left(\sum_{i=0}^{l-1} 2^i k_{x_i}\right)\right]} =$$

$$F^{l-1)}(\kappa_y, \nu_y, 0\ ,\ \kappa_x, \nu_x, 0) +$$

$$F^{l-1)}(\kappa_y, \nu_y, 0\ ,\ \kappa_x, \nu_x, 1) \cdot \underbrace{W_2^{n_{x_{l-1}}}}_{1\ \text{ó}\ -1} \cdot \underbrace{W_{2^l}^{\left(\sum_{i=0}^{l-2} 2^i k_{x_i}\right)}}_{twiddle_{\kappa_x}} +$$

$$F^{l-1)}(\kappa_y, \nu_y, 1\ ,\ \kappa_x, \nu_x, 0) \cdot \underbrace{W_2^{n_{y_{l-1}}}}_{1\ \text{ó}\ -1} \cdot \underbrace{W_{2^l}^{\left(\sum_{i=0}^{l-2} 2^i k_{y_i}\right)}}_{twiddle_{\kappa_y}} +$$

$$F^{l-1)}(\kappa_y,\nu_y,1\,,\,\kappa_x,\nu_x,1)\cdot\underbrace{W_2^{n_{y_{l-1}}}}_{1\,\acute{o}\,-1}\cdot\underbrace{W_2^{n_{x_{l-1}}}}_{1\,\acute{o}\,-1}\cdot twiddle\kappa_y\cdot twiddle_{\kappa_x} \tag{3.10}$$

La transformación completa requiere r de estas etapas. En cada etapa dos índices binarios, $n_{y_{r-l}}$ y $n_{x_{r-l}}$ se reemplazan por $k_{y_{l-1}}$ y $k_{x_{l-1}}$. Lo que antes era válido para *nodos duales*, será ahora aplicable a grupos de cuatro nodos. Aunque los factores *twiddle* parecen diferentes a los que se llegó para (3.3), ambos son equivalentes: dependiendo de si sacamos progresivamente los n y dejamos el sumatorio en κ, o al revés, tendremos los *twiddles* más homogéneos en las primeras etapas, o en las últimas.

La ecuación previa, 3.10, muestra los cuatro nodos en $F^{l-1)}$ a considerar para computar un elemento genérico de $F^{l)}$, cuando consideramos las cuatro combinaciones posibles de $n_{y_{r-l}}$ y $n_{x_{r-l}}$, obtenemos la mariposa completa que relaciona cuatro nodos a la salida con cuatro nodos en la entrada:

$$F^{l)}(0,\kappa_y,\nu_y\,,\,0,\kappa_x,\nu_x)=\underbrace{F^{l-1)}(\kappa_y,\nu_y,0\,,\,\kappa_x,\nu_x,0)}_{A}+$$
$$\underbrace{F^{l-1)}(\kappa_y,\nu_y,0\,,\,\kappa_x,\nu_x,1)}_{B}\cdot twiddle_{\kappa_x}+\underbrace{F^{l-1)}(\kappa_y,\nu_y,0\,,\,\kappa_x,\nu_x,1)}_{C}\cdot twiddle_{\kappa_y}+$$
$$\underbrace{F^{l-1)}(\kappa_y,\nu_y,0\,,\,\kappa_x,\nu_x,1)}_{D}\cdot twiddle_{\kappa_y}\cdot twiddle_{\kappa_x}; \tag{3.11a}$$

$$F^{l)}(0,\kappa_y,\nu_y\,,\,1,\kappa_x,\nu_x)=A-\underbrace{B\cdot twiddle_{\kappa_x}}_{B^*}+$$
$$\underbrace{C\cdot twiddle_{\kappa_y}}_{C^*}-\underbrace{D\cdot twiddle_{\kappa_y}\cdot twiddle_{\kappa_x}}_{D^*}; \tag{3.11b}$$

$$F^{l)}(1,\kappa_y,\nu_y\,,\,0,\kappa_x,\nu_x)\;=\;A+B^*-C^*-D^*; \tag{3.11c}$$

$$F^{l)}(1,\kappa_y,\nu_y\,,\,1,\kappa_x,\nu_x)\;=\;A-B^*-C^*+D^* \tag{3.11d}$$

Las ecuaciones a las que hemos llegado suponen el cómputo a realizar en los *kernels* de mariposa. Se muestran resumidamente en la figura 3.5. Es la extensión 2D de la subfigura 3.3.*b)*.

Para poder implementar estas nuevas mariposas de 4+4 nodos, el hardware subyacente ha de soportar la extensión conocida como *Multiple Render Targets*, (MRT) [OpenGL ARB 04], que es la que permite que un *kernel* dé

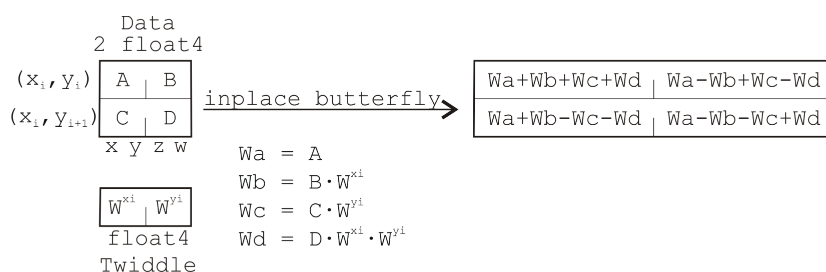


Figura 3.5: *Strels* involucrados en una mariposa Pease radix-2–radix-2. Los cuatro números complejos involucrados en el cómputo se representan por `A`, `B`, `C` y `D`. Se almacenan en dos *strels* adyacentes verticalmente: (x_i, y_i) y (x_i, y_{i+1}). Para llevar a cabo la mariposa *in-place* se requiere un *strel* extra que contenga los factores *twiddle* apropiadamente precomputados. El cómputo sigue las ecuaciones (3.11). Los resultados se depositan en dos *strels* con la disposición mostrada a la derecha.

salida a más de un *stream* simultáneamente. Por ejemplo, la arquitectura *G80* permite escribir hasta en cuatro. Las mariposas a las que hemos llegado se alimentan con dos *strels* de datos `float4` a la entrada, y vierten su salida igualmente a dos *strels* `float4`, por lo que estamos usando la extensión, pero sin aún explotarla al máximo.

La figura 3.6 muestra la combinación de *steams*, *kernels* y operadores de dominio que permiten implementar la FFT 2D propuesta. A este nivel, el mapeo es más parecido al propuesto para secuencias 1D sobre *streams* 2D (subsección 3.6) que al de FFT 2D via transformadas 1D y transposiciones.

En la subfigura **a)**, se ilustran los movimientos de datos que se darán por la recombinación $lsb_{x/y} \longleftrightarrow MSB_{x/y}$ propia de la variante Pease *radix-2–radix-2*.

La frontera de adyacencia lsb_x cae dentro de un *strel*, se da a nivel de *slot*. La vecindad lsb_y, en cambio, hay que buscarla a nivel de *strel*. Estas distancias marcan las posiciones donde hay que buscar los cuatro números complejos que entran en una mariposa *in-place*: un bloque o agrupación lógica de dos *strels* situados en filas consecutivas del *stream*. Así, en el esquema se observa un bloque lógico `A-B-C-D`, situado en una columna par, y otro bloque lógico formado por `H-I-J-K` en una columna inicialmente impar.

Las distancias MSB_x y MSB_y dividen horizontal y verticalmente al *stream* en cuartos, que designaremos `1q`, ..., `4q`. Hemos de procurar que los elementos que componen un bloque lógico finalicen distanciados MSB_x y MSB_y entre sí.

Para ello, tras la fase de cómputo *in-place* se han de combinar pares de *strels* transformados en columnas pares (que suministrarán los elementos que

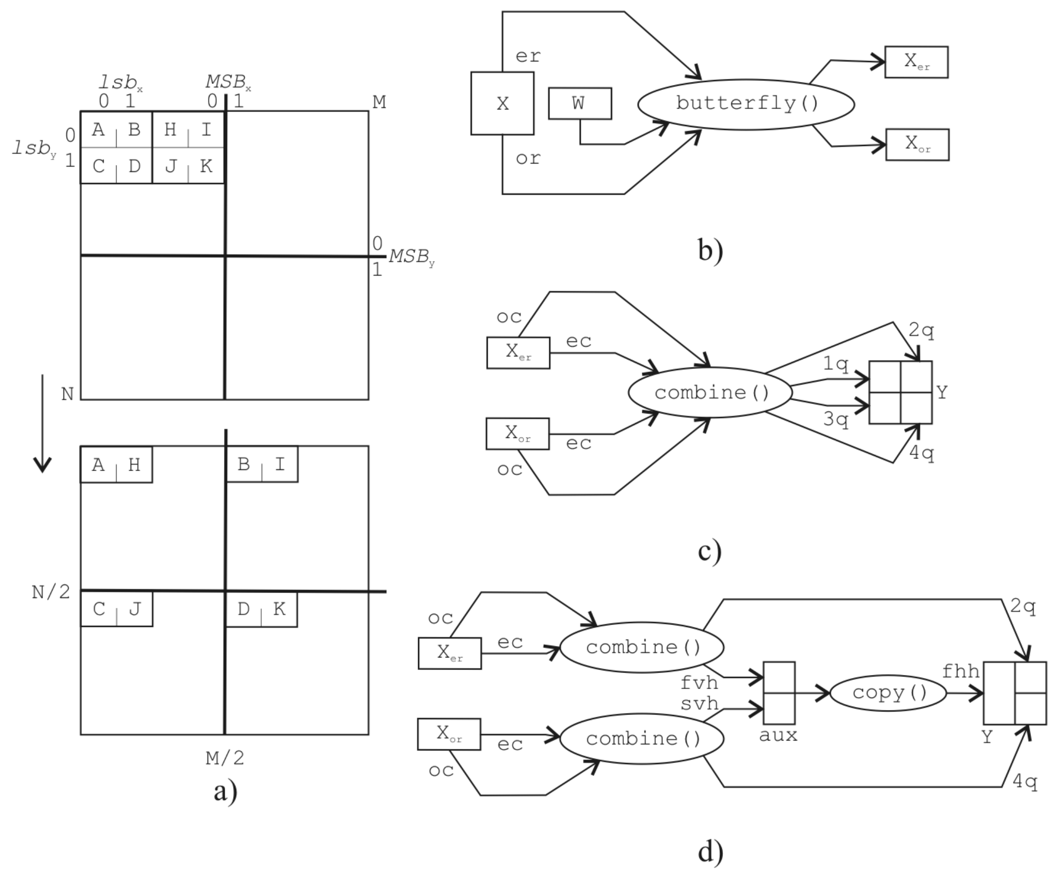


Figura 3.6: FFT 2D Pease *radix-2–radix-2* en términos de *streams* y *kernels*.

ocupen los *slots* `.xy` de salida) e impares (cuyos datos irán a los *slots* `.zw`). Nótense las sutiles diferencias con respecto a la subfigura 3.4.*a)*.

En la subfigura **b)** se muestra lo que se adelantó en la figura 3.5, el *kernel* que computa las mariposas *in-place* se alimenta de tres *strels*. Los cuatro datos complejos provienen de dos *strels* lsb_y-adyacentes. Para ello, en la llamada al *kernel* se usarán operadores de dominio que dividen el *stream* `X` en filas pares/impares (***e****ven*/***o****dd* ***r****ows*). Así los *strels* que contienen a `A-B` y `C-D` mapean sobre la misma posición, al igual que sucede con los pares `H-I` y `J-K`.

Los cuatro números complejos resultantes de cada instancia de mariposa *in-place* recaerán en sendos *strels*, sobre *streams* de tamaño mitad, que llamaremos $\texttt{X}_{\texttt{er}}$ y $\texttt{X}_{\texttt{or}}$. La extensión *MRT* es necesaria para poder realizar todos los cómputos en un único *kernel*.

En la subfigura **c)** se muestra cómo los *kernels* de combinación no realizan cómputo, pero son necesarios para redistribuir adecuadamente los datos conforme al esquema en *a)*. A la entrada toman cuatro *strels* correspondientes a dos bloques lógicos, los combinan y llevan a cuatro *strels*, cada uno en una porción del *stream* `Y` de salida (que jugará nuevamente el papel de `X` en la próxima etapa). Para ello, los dos *streams* a la entrada, $\texttt{X}_{\texttt{er}}$ y $\texttt{X}_{\texttt{or}}$, se han de volver a dividir por columnas pares/impares (***e****ven*/***o****dd* ***c****olumns*), mientras que al *stream* de salida se le han de aplicar los adecuados operadores de dominio para que cuente como 4 *substreams*.

En la subfigura **d)** se resalta el hecho de que el esquema en *c)*, tal cual se ha descrito, no es posible puesto que la extensión *MRT* no permite la escritura simultanea a diferentes dominios de un mismo *stream*. Por lo que se propone un esquema alternativo. La escritura a los 4 cuartos de `Y` se reemplaza con dos pasadas del *kernel* de combinación sobre uno y otro *stream* $\texttt{X}_{\texttt{er}}$ y $\texttt{X}_{\texttt{or}}$. Los resultados se vuelcan a a la segunda mitad horizontal de `Y`, y a un *stream* auxiliar. Una posterior llamada al *kernel* `copy()` pasa esos datos a la mitad horizontal que faltó por escribir en `Y`.

Esta combinación de *streams* y *kernels* se puede entender mejor ayudándonos de los esquemas 3.7 y 3.8. En la primera de estas figuras, 3.7, se muestra un *stream* que contiene una secuencia de 4×4 datos complejos, y cómo serán los cambios que produzca una etapa de transformación parcial: se muestra con distintos colores cómo se espera que se agrupen lógicamente los nodos a la entrada a las mariposas, sobre el *stream* `X`, y cómo se han de separar a la salida, sobre `Y`. Las operaciones de dominio que se esquematizan en 3.6.*c)* producen el mapeo ilustrado en 3.8, agrupando de tal manera los datos que permiten la correcta ejecución de ambos *kernels* (el de mariposa, en 3.6.**a)**, y de recombinación, en 3.6.**b)**). En esta figura, más que en ninguna otra, se hace notoria la relación entre las operaciones de dominio, el mapeo

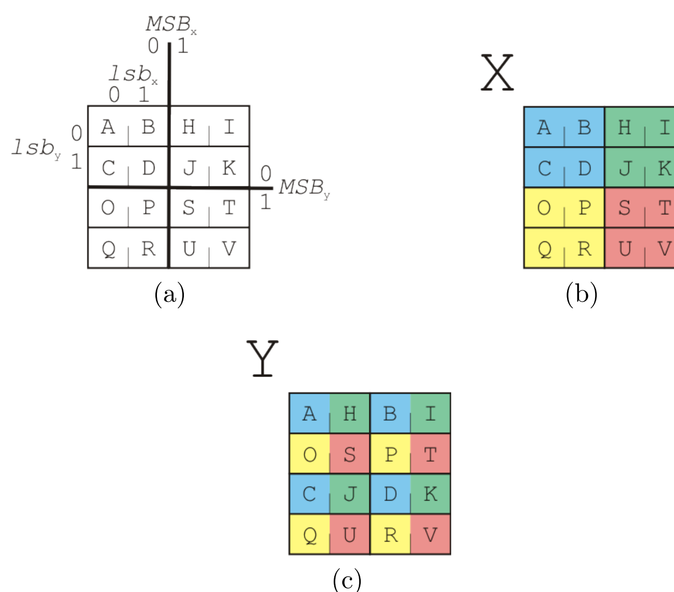


Figura 3.7: Disposición de una secuencia de 4×4 números complejos (`A, B, ..., V`) empaquetados de dos en dos sobre un *stream* 2D de tipo `float4` y tamaño 4×2. En **a)** se denotan las fronteras de bit menos y más significativo en cada dimensión, las cuales marcan las vecindades de datos que forman parte de una operación de mariposa a la entrada, y la disposición que han de tomar en la secuencia de salida. En **b)** y **c)** se ilustran con distintos colores —en *b)* sobre el *stream* `X` de entrada, y en *c)* sobre el *stream* `Y` de salida— los cuatro bloques de datos que participan en las cuatro mariposas por etapa que requiere un problema de este tamaño.

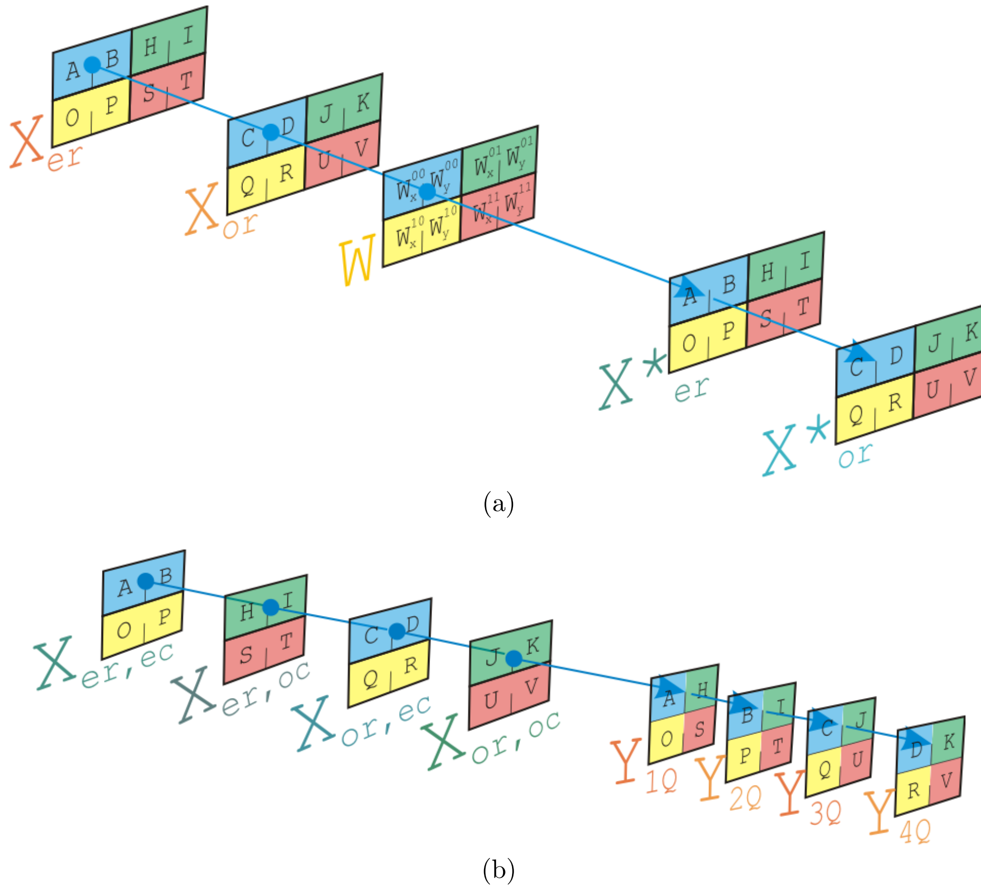
de *streams* y el paralelismo a nivel de datos.

3.7.1. Caso bidimensional, $M! = N$

Podemos ahora extender las fórmulas propuestas para el caso $N \times N$ al más general $N \times M$.

Esto será de utilidad en el caso de trabajar con números reales. En esa situación se puede reducir el espacio consumido en memoria, y el tiempo de ejecución, si en vez de trabajar con números complejos de fase nula, se opera con $N \times N/2$ números complejos y se aplican los cambios menores que la hacen equivalente a la transformada real del tamaño doble, como se detalla en, por ejemplo, Proakis and Manolakis [Proakis 96].

Las fórmulas de los factores *twiddle* varían ligeramente en este caso. El otro cambio lo encontraremos en la naturaleza diferenciada de las etapas de



(a)

(b)

Figura 3.8: Mapeo de las fases de cómputo y recombinacón sobre el *stream* de ejemplo de la figura 3.7. En **a)** se procede a dividir en filas pares (`er`) e impares (`or`) al *stream* `X` de entrada para alimentar, conjuntamente con el *stream* `W` de *twiddles* precomputados, el *kernel* de mariposa. La salida parcial de la fase de cómputo va a dos *streams*: $\mathtt{X}^*_{\mathtt{er}}$ y $\mathtt{X}^*_{\mathtt{or}}$. En la fase de recombinación, **b)**, estas salidas parciales se vuelven a dividir por columnas pares/impares (`ec`/`oc`), dando lugar a las cuatro entradas al *kernel* de combinación. Redistribuyendo los datos adecuadamente sobre sus cuatro cuadrantes `1q, ..., 4q` se obtiene la salida `Y` deseada (subfigura 3.7c).

transformación. Primero, los datos se transformarán 1D exclusivamente a lo largo de la dimensión de mayor tamaño (empleando para ello el método 1D FFT multifila). Esta parte del algoritmo finalizará cuando el número de índices binarios por transformar en la mayor de las dimensiones se iguale al número de índices en que se descompone la dimensión de menor tamaño. A partir de ahí se aplica el método recién explicado que colapsa dos índices por etapa.

Veamos las fórmulas de la mariposa para una secuencia que consideraremos de tamaño $N = 2^r$, en vertical, pero $M = 2^{r+q}$, en horizontal, con $q = log_2(M) - r > 0$. Esto es, la dimensión horizontal será 2^q veces mayor que la vertical.

En la primera fase del algoritmo, `l := 1...q`, los índices en vertical permanecen sin transformar:

$$F^{l)}(\underbrace{n_y}_{r\text{ bits}}, \underbrace{k_{x_{l-1}} \overbrace{k_{x_{l-2}}..k_{x_0}}^{\kappa_{0_x}:\, l-1\text{ bits}} \overbrace{n_{x_0}..n_{x_{r+q-l-1}}}^{\nu_{0_x}:\, r+q-l\text{ bits}}}_{r+q\text{ bits}}) =$$

$$\sum_{n_{x_{r+q-l}}}^{0,1} F^{l-1)}(n_y\ ,\ \kappa_{0_x}\nu_{0_x}n_{x_{r+q-l}}) \cdot W_{2^l}^{n_{x_{r+q-l}}\left(\sum_{i=0}^{l-1} 2^i k_{x_i}\right)} \tag{3.12}$$

Cuando resten r índices binarios en horizontal, se transforman simultáneamente con los verticales. La siguiente fórmula se aplica para `l := q+1...q+r`:

$$F^{l)}(k_{y_{l-q-1}} \overbrace{k_{y_{l-q-2}}..k_{y_0}}^{\kappa_y:\, l-1-q\text{ bits}} \overbrace{n_{y_0}..n_{y_{r-l-1+q}}}^{\nu_y:\, r-l+q\text{ bits}},$$

$$k_{x_{l-1}} \underbrace{\overbrace{k_{x_{l-2}}..k_{x_q}}^{l-1-q\text{ bits}} \overbrace{k_{x_{q-1}}..k_{x_0}}^{q\ bits}}_{\kappa_x:\, l-1\text{ bits}} \underbrace{n_{x_0}..n_{x_{r+q-l-1}}}_{\nu_x:\, r+q-l\text{ bits}}) =$$

$$\sum_{n_{y_{r-l+q}}}^{0,1} \sum_{n_{x_{r+q-l}}}^{0,1} F^{l-1)}(\kappa_y\nu_y\, n_{y_{r-l+q}}\ ,\ \kappa_x\nu_x\, n_{x_{r+q-l}})$$

$$\cdot\, W_{2^l}^{\left[2^q n_{y_{r-l+q}}\left(\sum_{i=0}^{l-q-1} 2^i k_{y_i}\right) + n_{x_{r+q-l}}\left(\sum_{i=0}^{l-1} 2^i k_{x_i}\right)\right]} \tag{3.13}$$

3.7.2. Consideraciones acerca de la variante Pease 2D sobre GPU

El esquema propuesto para llevar a cabo la FFT 2D[2] tiene las siguientes implicaciones al implementarse sobre una GPU *stream*-programada:

- los *kernels* de mariposa, que previamente operaban sobre 1 *strel* a la entrada y a la salida —y con eso les bastaba para contemplar los 2 nodos duales que participan en una FFT 1D—, los hemos aumentado para que ahora den cuenta de los 4 números complejos que forman la mariposa 2D. Aquí es donde juega su papel la extensión *MRT*, permitiéndonos aumentar la relación ente operaciones aritméticas y accesos a memoria. Como se ha adelantado, no pararemos aquí. A continuación el método se extiende y se hace lo suficientemente general como para atender a mariposas de tamaño 8 (copando así el límite actual de 4 *streams* de salida) o superior (en previsión de mejoras en las arquitecturas GPU).

- el número de etapas requerido por un problema de tamaño $N \times N$ será $log_2(N)$, donde los métodos *row-column* empleaban $2log_2(N)$ etapas. Por lo tanto, aunque ahora los cómputos son más elaborados y requieren *kernels* de más instrucciones, mientras aumentamos el número de nodos por mariposa desciende el número de llamadas a *kernel*, lo cual nos beneficia.

- ya no son necesarias las transposiciones. Se han entreverado con el resto del método.

- las recombinaciones tras el cómputo —costosas en términos de llamadas a *kernel* y por reducir la relación cómputo/accessos—, permancecen como el coste a pagar por usar la variante de Pease. Pero al menos en esta variante se delimita claramente lo que es cómputo de lo que son movimientos de datos. Esta división es clave para poder seguir extendiendo el método, con lo que su sobrecoste se justifica cuanto más complicado se vuelve éste. Además la recombinación también se beneficia, aunque en menor medida, del uso de *MRT*.

En el apéndice A.1 se recoge una selección representativa de código real que implementa esta última FFT 2D $N \neq M$. Es el más elaborado de los métodos hasta este punto, y guarda un cierto equilibrio entre la complejidad

[2]A continuación se extenderá para que cope los 4 *streams* de salida que la extensión *MRT* permite. Pero la discusión es igualmente pertinente en ese caso, por lo que se adelanta hasta aquí, donde el esquema aún permanece «razonablemente» claro.

del código y el número de técnicas e ideas que éste contiene (la próxima implementación es notablemente más eficiente, pero también más complicada). Muy diferente a como se expresaría el mismo algoritmo en un lenguaje secuencial de alto nivel, el código resultante es bastante reducido, lo que refleja la alta expresividad que permite el lenguaje *BrookGPU*. Por último, enfatizar que su uso como lenguaje no es imperativo, solo un medio para poder plantear un algoritmo pleno de significado en el paradigma de *stream*.

3.8. Caso bidimensional, descomposición 4-2–2-4

Para incrementar aún más la intensidad aritmética, reduciendo el número de veces que cada dato se trae de memoria e incrementando el número de operaciones en que participa una vez está disponible, desarrollaremos un esquema de FFT 2D que extiende la variante de Pease, de tal manera que cada mariposa implique al mayor número de nodos posible. Actualmente ese límite viene impuesto por la extensión *MRT*: 1 *kernel* puede dar salida hasta 4 *streams* de tipo `float4`. Estos 4 *strels* pueden almacenar 8 números complejos, y por tanto, ése será el tamaño de mariposa que busquemos.

En el caso 1D la obtendríamos directamente descomponiendo en base octal el índice de transformación (obtendríamos un método *radix-8*). Que sería extrapolable a 2D con solo recurrir a la propiedad de separabilidad: ver sección 3.5. Pero resulta más efectivo trabajar simultáneamente en ambas dimensiones operando con un enfoque de *bases compuestas*. Esto es, descomponiendo la dimension horizontal en una base distinta a la empleada en la vertical, pero colapsando ambas simultáneamente.

Si usáramos base 2 para la dimensión y y base 4 para la dimensión x obtendríamos mariposas de 8 nodos, pero con el problema de que solo podríamos computar FFTs de tamaño $2^r \times 4^r$. Esto lo subsanaremos haciendo que las descomposiciones no solo sean en bases compuestas, sino además *alternadas*. Por ejemplo, si a cada etapa que colapse un índice binario en x y uno cuaternario en y, le sigue una etapa binaria en y y cuaternaria en x, el desbalance que se produce en una etapa por tener una base mayor en una que en otra dimensión, observado globalmente se compensa, permitiendo problemas de tamaño $2^r \times 2^r$. Si ampliamos la libertad, no contrarrestando los desbalances cada dos etapas, podremos operar en cualquier tamaño $2^r \times 2^q$.

Como además en la formulación no nos ceñiremos a bases 2 y 4, aunque sean las únicas necesarias para lograr mariposas de 8 nodos, el método obtenido es lo suficientemente general como para que, de cambiar este límite, el

marco teórico siga siendo aplicable.

3.8.1. Descomposición en bases genéricas

Sustituyamos la función de índice de base binaria en (3.2) por:

$$n_x = I(n_{x_{r-1}}, n_{x_{r-2}}, \ldots, n_{x_1}, n_{x_0}) = \prod_{i=0}^{r-2} d_{x_i} n_{r-1} + \prod_{i=0}^{r-3} d_{x_i} n_{r-2} + \cdots + d_{x_0} n_{x_1} + n_{x_0} \tag{3.14}$$

capaz de expresar un índice n_x en la base de coeficientes indeterminados $\{d_{x_i}\}$. Por supuesto (3.14) es equivalente a (3.2) para el caso $d_{x_i} = 2$, para $i = 0, \ldots, r-1$.

De esta manera es posible expresar la expansión del sumatorio de la etapa l de la variante de Pease para una secuencia 2D, en la cual los índices $n_{y_{r-l}}$ y $n_{x_{r-l}}$ se transformen respectivamente en $k_{y_{l-1}}$ y $k_{x_{l-1}}$, y donde n_x y k_x, y, n_y y k_y, han sido descompuestos en bases genéricas $\{d_{x_i}\}$ y $\{d_{y_i}\}$

$$\begin{aligned}
F^{l)}(k_{y_{l-1}}, \overbrace{k_{y_{l-2}}, .., k_{y_0}}^{\kappa_y:\, l-1\ digitos}, \overbrace{n_{y_0}, .., n_{y_{r-l-1}}}^{\nu_y:\, r-l\ digitos}, \quad k_{x_{l-1}}, \overbrace{k_{x_{l-2}}, .., k_{x_0}}^{\kappa_x}, \overbrace{n_{x_0}, .., n_{x_{r-l-1}}}^{\nu_x}) = \\
\sum_{n_{y_{r-l}}=0}^{d_{y_{r-l}}-1} \sum_{n_{x_{r-l}}=0}^{d_{x_{r-l}}-1} F^{l-1)}(\kappa_y,\ \nu_y,\ n_{y_{r-l}}\ ,\ \ \kappa_x,\ \nu_x,\ n_{x_{r-l}}) \\
\cdot W_{\prod_{i=0}^{l-1} d_{y_i}}^{n_{y_{r-l}} \sum_{i=0}^{l-1}\left(\prod_{j=0}^{i-1} d_{y_j} k_{y_i}\right)} \cdot W_{\prod_{i=0}^{l-1} d_{x_i}}^{n_{x_{r-l}} \sum_{i=0}^{l-1}\left(\prod_{j=0}^{i-1} d_{x_j} k_{x_i}\right)} = \\
\sum_{n_{y_{r-l}}=0}^{d_{y_{r-l}}-1} \sum_{n_{x_{r-l}}=0}^{d_{x_{r-l}}-1} F^{l-1)}(\kappa_y,\ \nu_y,\ n_{y_{r-l}}\ ,\ \ \kappa_x,\ \nu_x,\ n_{x_{r-l}}) \\
\cdot W_{d_{y_{l-1}}}^{n_{y_{r-l}} k_{y_{l-1}}} \cdot \underbrace{\left[W_{\prod_{i=0}^{l-1} d_{y_i}}^{\sum_{i=0}^{l-2}\left(\prod_{j=0}^{i-1} d_{y_j} k_{y_i}\right)}\right]^{n_{y_{r-l}}}}_{twiddle_{\kappa_y}} \cdot W_{d_{x_{l-1}}}^{n_{x_{r-l}} k_{x_{l-1}}} \cdot \underbrace{\left[W_{\prod_{i=0}^{l-1} d_{x_i}}^{\sum_{i=0}^{l-2}\left(\prod_{j=0}^{i-1} d_{x_j} k_{x_i}\right)}\right]^{n_{x_{r-l}}}}_{twiddle_{\kappa_x}}
\end{aligned} \tag{3.15}$$

Esta ecuación es completamente genérica. Es solo cuando volvemos sobre nuestro objetivo de maximizar la extensión *MRT* que imponemos que los índices se han de descomponer con $d_{x_l}, d_{y_l} \in \{2, 4\}$ y $d_{x_l} \cdot d_{y_l} \leq 8$. Nótese que estas condiciones no descartan el caso de etapas con coeficientes binarios coincidentes: radix-2–radix-2. O incluso casos donde se transforma solo a lo largo de una etapa. Las combinaciones posibles d_{y_l}–d_{x_l} en una etapa bajo estas premisas son: (4–2, 2–4, 2–2, –4, –2).

Un problema de tamaño 16×128 requeriría el uso de todas ellas:

$$\begin{array}{l} B_y := \{\ 4\ \ 2\ \ 2\ \ -\ \ -\ \} \\ B_x := \{\ 2\ \ 4\ \ 2\ \ 4\ \ 2\ \} \end{array}$$

Para lograr la completa generalización del método hemos implementado, en términos de *streams* y *kernels*, a partir de la ecuación (3.15), todas las combinaciones descritas. Para obtener la máxima eficiencia usaremos bases que combinen pares de etapas 4–2, 2–4, mientras el tamaño que reste por transformar sea mayor que 8×8. Solo entonces se usarán el resto de combinaciones.

Veámos solamente la implementación del caso $d_{y_{r-l}} = 4$ y $d_{x_{r-l}} = 2$. Para esta combinación la ecuación (3.15) se transforma en:

$$\begin{aligned} &F^{l)}(k_{y_{l-1}}, \kappa_y, \nu_y, \quad k_{x_{l-1}}, \kappa_x, \nu_x) = \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 0\,, \ \kappa_x, \nu_x, 0) + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 0\,, \ \kappa_x, \nu_x, 1) \cdot twiddle_{\kappa_x} \cdot W_2^{k_{x_{l-1}}} + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 1\,, \ \kappa_x, \nu_x, 0) \cdot twiddle_{\kappa_y} \cdot W_4^{k_{y_{l-1}}} + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 1\,, \ \kappa_x, \nu_x, 1) \cdot twiddle_{\kappa_y} \cdot W_4^{k_{y_{l-1}}} \cdot twiddle_{\kappa_x} \cdot W_2^{k_{x_{l-1}}} + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 2\,, \ \kappa_x, \nu_x, 0) \cdot twiddle_{\kappa_y}{}^2 \cdot W_4^{2k_{y_{l-1}}} + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 2\,, \ \kappa_x, \nu_x, 1) \cdot twiddle_{\kappa_y}{}^2 \cdot W_4^{2k_{y_{l-1}}} \cdot twiddle_{\kappa_x} \cdot W_2^{k_{x_{l-1}}} + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 3\,, \ \kappa_x, \nu_x, 0) \cdot twiddle_{\kappa_y}{}^3 \cdot W_4^{3k_{y_{l-1}}} + \\ &\quad F^{l-1)}(\kappa_y, \nu_y, 3\,, \ \kappa_x, \nu_x, 1) \cdot twiddle_{\kappa_y}{}^3 \cdot W_4^{3k_{y_{l-1}}} \cdot twiddle_{\kappa_x} \cdot W_2^{k_{x_{l-1}}} \end{aligned} \tag{3.16}$$

Donde tenemos los ocho elementos de $F^{l-1)}$, resultantes de colapsar ambos índices de sumatorio $n_{y_{r-l}}$ y $n_{x_{r-l}}$, a considerar en el cómputo de un elemento de la secuencia $F^{l)}$ con $k_{y_{l-1}}$ y $k_{x_{l-1}}$ indeterminados. Cuando también tenemos en consideración los índices en k, obtenemos la mariposa que relaciona ocho nodos a la entrada con ocho nodos a la salida, como sigue:

$$\begin{aligned} &F^{l)}(0, \kappa_y, \nu_y\,, \ 0, \kappa_x, \nu_x) = \\ &\quad \overbrace{F^{l-1)}(\kappa_y, \nu_y, 0\,, \ \kappa_x, \nu_x, 0)}^{A} + \\ &\quad \overbrace{F^{l-1)}(\kappa_y, \nu_y, 0\,, \ \kappa_x, \nu_x, 1) \cdot twiddle_{\kappa_x}}^{B} + \end{aligned}$$

$$\overbrace{F^{l-1)}(\kappa_y,\ \nu_y,\ 1\ ,\ \ \kappa_x,\ \nu_x,\ 0)\cdot twiddle_{\kappa_y}}^{C}+$$
$$\overbrace{F^{l-1)}(\kappa_y,\ \nu_y,\ 1\ ,\ \ \kappa_x,\ \nu_x,\ 1)\cdot twiddle_{\kappa_y}\cdot twiddle_{\kappa_x}}^{D}+$$
$$\overbrace{F^{l-1)}(\kappa_y,\ \nu_y,\ 2\ ,\ \ \kappa_x,\ \nu_x,\ 0)\cdot {twiddle_{\kappa_y}}^2}^{E}+$$
$$\overbrace{F^{l-1)}(\kappa_y,\ \nu_y,\ 2\ ,\ \ \kappa_x,\ \nu_x,\ 1)\cdot {twiddle_{\kappa_y}}^2\cdot twiddle_{\kappa_x}}^{F}+$$
$$\overbrace{F^{l-1)}(\kappa_y,\ \nu_y,\ 3\ ,\ \ \kappa_x,\ \nu_x,\ 0)\cdot {twiddle_{\kappa_y}}^3}^{G}+$$
$$\overbrace{F^{l-1)}(\kappa_y,\ \nu_y,\ 3\ ,\ \ \kappa_x,\ \nu_x,\ 1)\cdot {twiddle_{\kappa_y}}^3\cdot twiddle_{\kappa_x}}^{H} \quad (3.17a)$$

$$F^{l)}(0,\kappa_y,\nu_y\ ,\ 1,\kappa_x,\nu_x) = (A-B)+(C-D)+(E-F)+(G-H); \quad (3.17b)$$
$$F^{l)}(1,\kappa_y,\nu_y\ ,\ 0,\kappa_x,\nu_x) = (A+B)+(C+D)\cdot(-i)+(E+F)\cdot(-1)+(G+H)\cdot i; \quad (3.17c)$$
$$F^{l)}(1,\kappa_y,\nu_y\ ,\ 1,\kappa_x,\nu_x) = (A-B)+(C-D)\cdot(-i)+(E-F)\cdot(-1)+(G-H)\cdot i; \quad (3.17d)$$
$$F^{l)}(2,\kappa_y,\nu_y\ ,\ 0,\kappa_x,\nu_x) = (A+B)+(C+D)\cdot(-1)+(E+F)+(G+H)\cdot(-1); \quad (3.17e)$$
$$F^{l)}(2,\kappa_y,\nu_y\ ,\ 1,\kappa_x,\nu_x) = (A-B)+(C-D)\cdot(-1)+(E-F)+(G-H)\cdot(-1); \quad (3.17f)$$
$$F^{l)}(3,\kappa_y,\nu_y\ ,\ 0,\kappa_x,\nu_x) = (A+B)+(C+D)\cdot i+(E+F)\cdot(-1)+(G+H)\cdot(-i); \quad (3.17g)$$
$$F^{l)}(3,\kappa_y,\nu_y\ ,\ 1,\kappa_x,\nu_x) = (A-B)+(C-D)\cdot i+(E-F)\cdot(-1)+(G-H)\cdot(-i); \quad (3.17h)$$

La traducción de esas mariposas en código de *kernel* que use el juego de instrucciones *4-way SIMD* de la GPU se presenta en el cuadro 3.2. La otra parte del problema es cómo llevar los datos apropiados a los *kernels* por medio de operaciones de *stream*, lo cual ilustramos con las figuras 3.9 y 3.10. La operativa es similar a la ya descrita para *radix-2–radix-2*.

Para las restantes combinaciones de d_{y_l}–d_{x_l}, se ha de proceder de manera análoga.

```
kernel void butterfly42(float4 AB<>, float4 CD<>,
                        float4 EF<>, float4 GH<>,
                        float4 twiddleXY<>,
                        out float4 ab<>, out float4 cd<>,
                        out float4 ef<>, out float4 gh<>) {
  float4 ABtwiddle, CDtwiddle, EFtwiddle, GHtwiddle;
  float4 p, q, r, s;
  float4 pr, qs, pmr, qmsi;
  float4 x0_x = float4(1, 0, twiddleXY.x, twiddleXY.y); // (1, tx)
  float4 y1_y1x = multComplex2(twiddleXY.zwzw, x0_x);   // (ty, tytx)
  float4 y2_y2x = multComplex2(twiddleXY.zwzw, y1_y1x); // (tyty, tytytx)
  float4 y3_y3x = multComplex2(twiddleXY.zwzw, y2_y2x); // (tytyty, tytytytx)
  float4 plusMinus = float4(1, 1, -1, -1);              // (1, -1)

  ABtwiddle = multComplex2(AB, x0_x);
  CDtwiddle = multComplex2(CD, y1_y1x);
  EFtwiddle = multComplex2(EF, y2_y2x);
  GHtwiddle = multComplex2(GH, y3_y3x);

  p = ABtwiddle.zwxy + ABtwiddle*plusMinus;
  q = CDtwiddle.zwxy + CDtwiddle*plusMinus;
  r = EFtwiddle.zwxy + EFtwiddle*plusMinus;
  s = GHtwiddle.zwxy + GHtwiddle*plusMinus;

  pr  = p + r;
  qs  = q + s;
  pmr = p - r;
  qmsi= (q.yxwz-s.yxwz)*plusMinus.xwyz;

  ab = pr  + qs;
  cd = pmr + qmsi;
  ef = pr  - qs;
  gh = pmr - qmsi;
}
```

Cuadro 3.2: Código *BrookGPU* del *kernel* que computa la mariposa de Pease *radix-4–radix-2* descrita por las ecuaciones (3.17).

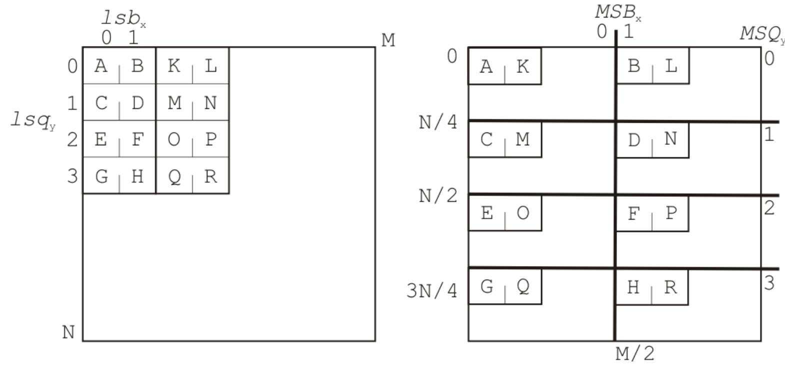


Figura 3.9: Distancias entre los digitos binarios más y menos significativos en horizontal, y los dígitos cuaternarios más y menos significativos en vertical, que determinan la agrupación de 8 nodos, a la entrada y a la salida, de dos mariposas de Pease *radix-4–radix-2*, sobre columnas par e impar, de un *stream* 2D de `float4`.

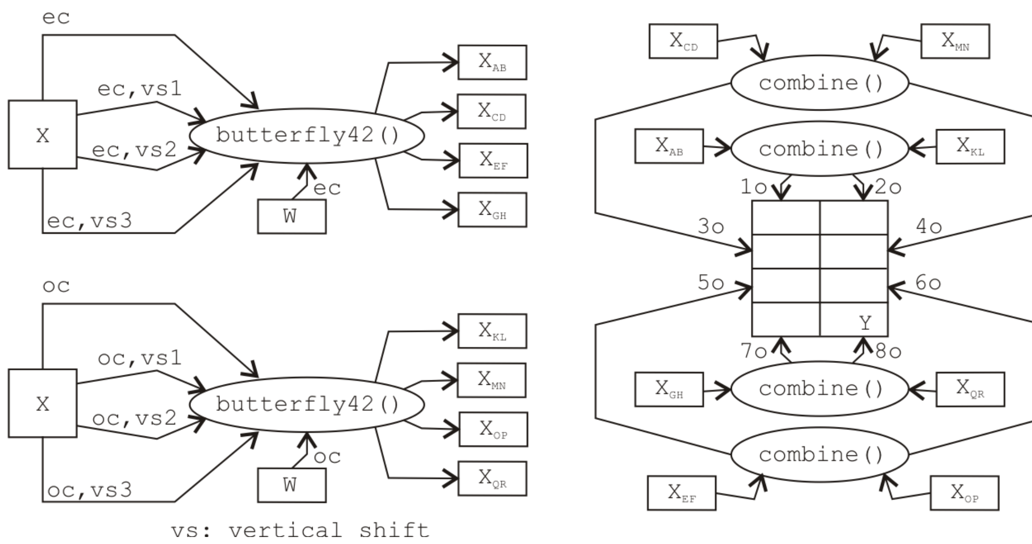


Figura 3.10: Operaciones de *stream* que realizan simultáneamente la transformación de un índice binario en horizontal y uno cuaternario en vertical.

	Tiempos de transferencia (ms)			
	PCI-Express v1.1		*PCI-Express v2.0 16x*	
↓ N	CPU → GPU	GPU → CPU	*CPU → GPU*	*GPU → CPU*
64^2	0.01 / **0.03**	0.05 / **0.11**	*0.03 / **0.04***	*0.08 / **0.09***
128^2	0.06 / **0.09**	0.23 / **0.25**	*0.07 / **0.13***	*0.10 / **0.15***
256^2	0.18 / **0.73**	0.39 / **0.72**	*0.27 / **0.65***	*0.25 / **0.48***
512^2	1.88 / **3.70**	1.29 / **2.33**	*1.43 / **2.70***	*0.80 / **1.59***
1024^2	7.90 / **15.9**	4.61 / **9.2**	*5.02 / **9.89***	*3.23 / **6.65***

Cuadro 3.3: Tiempos de transferencia de datos CPU–GPU sobre PCI-Express v1 y *v2*. (datos reales/**datos complejos**)

3.9. Implementación y rendimiento

Los siguientes resultados se obtuvieron usando BrookGPU (version 0.4, CVS actualizado a Marzo 2006), con el entorno de ejecución OpenGL, sobre una plataforma Windows XP (no existe soporte para la extensión *MRT* en la versión Linux de *BrookGPU*). Se usaron dos tarjetas distintas: una GeForce 6800 Ultra (con un motor NVIDIA nv40) y una GeForce 7800 GTX (motor G70) conectada a la máquina host a traves de un *slot* PCI-Express con una placa madre dotada de un *chipset* nForce4. La versión del driver gráfico fue la 81.98 (actualizada a Febrero de 2006). Para el mejor de nuestros algoritmos se dan también los tiempos obtenidos con una GeForce 8800 GTX (motor G80). En este caso el host cambia: será un equipo con chipset nForce 690i, PCI-Express 2.0, corriendo GNU/Linux.

Rendimiento observado

Los cuadros 3.3 y 3.4 muestran los tiempos requeridos para ejecutar la FFT sobre GPU para problemas de diverso tamaño. Los tiempos de transferencia entre GPU y CPU se aislan de los tiempos de cómputo. Si bien hay que tener en cuenta que penalizan el uso de la GPU como coprocesador. En cambio, en algunas aplicaciones en que los datos no han de pasar por la CPU se pueden ignorar.

Los tiempos de cómputo se recogen en el cuadro 3.4. El rendimiento obtenido es similar, independientemente de la organización de los datos, para las tres primeras versiones, las no-*MRT*. El tiempo requerido para ejecutar una FFT 1D de tamaño N^2 dobla el de ejecutar las N FFTs 1D de tamaño N, porque se realizan el doble de etapas. Las diferencias de tiempo entre la 1D N^2 y la FFT 2D no-*MRT* se debe a las trasposiciones.

En cambio el rendimiento que se obtiene, para tamaños de problema 512

	Tiempos de cómputo (ms)			
tipo de FFT:	N 1D N		1D N^2	
GPU:	6800	7800	6800	7800
size, N:64	0.70 / **0.91**	0.67 / **0.85**	1.67 / **1.83**	1.75 / **1.90**
128	0.97 / **1.26**	0.86 / **1.18**	2.00 / **2.28**	2.1 / **2.30**
256	1.48 / **2.75**	1.25 / **1.97**	2.43 / **4.60**	2.6 / **3.15**
512	4.8 / **9.80**	3.2 / **6.60**	9.2 / **18.9**	6.2 / **12.8**
1024	18.2 / **39.8**	12.4 / **27.3**	38.5 / **81.0**	26.3 / **56.2**
tipo de FFT:	1D+1D $N \times N$		2D $N \times N$	
GPU:	6800	7800	6800	7800
size, N:64	1.63 / **1.85**	1.60 / **1.74**	0.80 / **0.88**	1.34 / **1.28**
128	2.04 / **2.04**	1.97 / **2.27**	1.00 / **1.12**	1.41 / **1.55**
256	3.08 / **5.3**	2.54 / **3.83**	1.39 / **2.69**	1.86 / **2.16**
512	10.1 / **20.7**	7.10 / **13.6**	5.7 / **10.9**	4.17 / **7.50**
1024	41.9 / **89.8**	27.9 / **58.4**	24.1 / **48.0**	17.17 / **32.66**

Cuadro 3.4: Tiempos de cómputo de las implementaciones de los distintos algoritmos FFT sobre una Geforce 6800 Ultra y una Geforce 7800 GTX, ambas conectadas a la CPU por un bus PCI-Express. Los tiempos no incluyen la transferencia inicial de los datos, ni las operaciones de *bit-reversal*. Los algoritmos, organizados por columnas, son: N FFTs 1D de tamaño N, FFT 1D de tamaño N^2, FFT 2D de tamaño $N \times N$ por *row-column*, y FFT 2D de tamaño $N \times N$ *radix-2–radix-2*. (datos reales / **datos complejos**)

	Tiempos de cómputo (ms)				
	D. Horn	ULL 22-KC	ULL 22-M41	ULL 4224	FFTW
↓ N	G7:re/**co**	G7:re/**co**	G7:re/**co**	**G7**/***G8***	CPU 3500+
64^2	0.58 / **0.61**	1.34 / **1.28**	0.90 / **0.91**	**1.58** ***1.43***	0.05 / **0.11**
128^2	0.67 / **0.85**	1.41 / **1.55**	1.12 / **1.01**	**1.69** ***1.63***	0.31 / **0.86**
256^2	1.30 / **2.42**	1.86 / **2.16**	1.29 / **1.54**	**1.78** ***1.80***	1.99 / **10.0**
512^2	4.6 / **9.0**	4.17 / **7.52**	3.6 / **6.5**	**4.98** ***4.72***	10.6 / **43.3**
1024^2	18.8 / **38.8**	17.17 / **32.66**	14.8 / **28.0**	**21.8** ***16.7***	52.3 / **196**

Cuadro 3.5: Tiempos de cómputo de la FFT 2D sobre secuencias de datos reales y **complejos** para las implementaciones libgpufft de D. Horn incluida en el proyecto BrookGPU-Stanford; nuestras versiones MRT *radix-2–radix-2*, (22-KC:Kernel copy, figura 3.6.*c)*) y (22-M41: *MRT* 4 a 1, figura 3.6.*d)*; nuestra versión que alterna *radix-4–radix-2* con *radix-2–radix-4* (4224); y la librería FFTW sobre GPU. En la columna 4224, ambos tiempos corresponden a secuencias de números complejos, a la izquierda ejecutando sobre una GeForce 7800 GTX, y a la derecha sobre una *GeForce 8800 GTX*. La ejecución sobre la 8800 se realizó en Linux.

y 1024, es el doble con la versión *MRT*, frente a la no-*MRT*. Dicho esto, el único beneficio de la versión 2D no-*MRT* es introducir y hacer más fácil de comprender la más elaborada versión *MRT*. Los tiempos que se dan para la versión *MRT* son los obtenidos al aplicar el esquema de la subfigura 3.6.*d)*, los cuales requieren un *kernel* adicional de copia respecto al esquema 3.6.*c)*, esto se aliviaría si la *MRT* permitiese la escritura sobre varios dominios del mismo *stream* de salida, o incluso si *BrookGPU* permitiese la copia directa entre zonas de memoria, en lugar de tener que recurrir a un *kernel* para lograrlo.

3.10. Comparativa con otros autores

La mayoría de trabajos de otros autores ponen más énfasis en la implementación (que aproveche las últimas extensiones aparecidas), que en la selección de un algoritmo adecuado a la arquitectura: optando por emplear la variante de Cooley&Tukey heredada de la CPU. Tan solo unos cuantos, entre ellos Jansen *et al.* y Horn, escogen otro algoritmo como punto de partida de sus implementaciones. Sucede que la discusión entre implementaciones, en lugar de realizarse a nivel algorítmico, está muy condicionado por el último modelo de tarjeta en el mercado. En cambio la discusión en torno al algoritmo sin más, puede a menudo resultar inútil, debido al salto no despreciable,

entre el algoritmo «ideal» y su implementación condicionada por la tarjeta gráfica subyacente, la versión de los drivers, el *slot* de conexión, e incluso el sistema operativo.

Ya hemos discutido el porqué de la elección de la variante de Pease: especialmente debido al mejor uso del ancho de memoria, al permitir mapeo directo de *streams*. Es el momento de reforzar este argumento:

- La variante escogida es capaz de encontrar los datos que entran a su mariposa por mapeo directo cuando se usan *streams* `float4`. Y es la única variante que lo permite: ni la variante de C&T, ni la de Stockham, ni la de Pease con decimación en frecuencia propician la agrupación *lsb* durante todas las etapas.

- Los operadores de dominio dividen los *streams* en zonas lógicas de tal manera que no son necesarias sentencias condicionales en los *kernels*. De este modo, ni se traen datos de memoria, ni se realizan cómputos que luego vayan a ser desechados, como ocurre con las implementaciones de C&T y Pease inversa[3]. Tan solo nos sucedía esto en las implementaciones no-*MRT*, e incluso así, era una pérdida sobre datos leídos, no sobre datos computados, y no requería la evaluación de ninguna condicional.

- Al aprovechar el empaquetado de los nodos relacionados en una mariposa, y de los factores *twiddle*, cada dato se lee una sola vez. No existen cómputos replicados en distintas zonas del *stream*.

- La información acerca de dónde leer y escribir los datos no es necesaria, por lo que ni se consulta, ni se computa en un *kernel*. La recombinación de los datos se logra exclusivamente por medio de operaciones a nivel de *stream*, lo que significa que se ejecutan a nivel del *rasterizador*, no en el procesador de fragmento. El trabajo se encuentra mejor repartido entre las unidades que conforman el *pipeline* de la GPU. En este sentido, la variante de C&T se ve perjudicada al necesitar información acerca de dónde traer sus datos: lo cual depende de la posición y de la etapa. Sumanaweera y Liu lograron rebajar este punto, haciendo uso de *display lists*, que permitieron eliminar la dependencia de la etapa: pero con la variante de Pease, directamente no existe tal dependencia. Aunque en su trabajo estos últimos autores indican que su implementación balancea la carga de trabajo entre el procesador de vértice y el de fragmento, su código no hace uso del primero: sino de *display lists* que

[3]Pease con DIF: es la elegida por D. Horn, a pesar de que en su artículo afirma usar una C&T.

agrupan bloques de vértices dentro de los cuales el rasterizador interpola las direcciones y los factores *twiddle*. Es de alguna manera similar al uso de operaciones de dominio en Pease, pero que tan solo alivia parcialmente un problema que con la elección de la variante adecuada simplemente deja de existir.

- Hemos usado la extensión *MRT* de un modo muy distinto a como la usan Sumanaweera y Liu. Ellos emplean *MRT* y el modo estéreo de renderizado (logrando 8 buffers de `float1`) para computar simultáneamente dos transformadas sobre secuencias independientes, donde la *MRT* solo se explota para reusar en ambas FFTs las direcciones de las que leer, y los *twiddle*.

En el cuadro 3.5 se muestra los tiempos obtenidos por nuestras implementaciones *MRT*, tanto *radix-2–radix-2*, como la que alterna descomposiciones 4-2–2-4, así como los de la implementación de D. Horn, incluida en el lenguaje BrookGPU, y los tiempos obtenidos con la librería FFTW. Estos últimos se obtuvieron para una CPU AMD 3500+ con 512KB de *cache-onchip*. Las versiones GPU se evaluaron con una Geforce 7800 GTX, y para la implementación 4-2–2-4, se incluyen adicionalmente los tiempos sobre una GeForce 8800 GTX.

Para la versión *MRT radix-2–radix-2* se consideran tanto la implementación del esquema 3.6.*d)* como el de la subfigura 3.6.*c)*, en la tercera y cuarta columna, respectivamente. Los tiempos requeridos son menores para la versión que escribe sobre varios dominios no solapados del mismo *stream* de salida, lo cual no está permitido a día de hoy por la extensión *MRT*. Incluso con ese *handicap*, nuestra otra implementación, consume un tercio del tiempo que requiere la FFTW para secuencias grandes de números reales, como las que lleva aparejado el problema del reenfoque fotografico digital, cuando se considera una tarjeta ya obsoleta como la 7800 GTX.

Para secuencias de complejos, los resultados son incluso mejores, reduciéndose para el tamaño 1024×1024 a un 17 %, cuando se ejecuta la versión *radix-2–radix-2*, y a un 11 % usando la de bases 4–2, 2–4 alternadas, de nuevo con la tarjeta 7800 GTX. En el caso de usar una más actual, la 8800 GTX, y nuestra mejor implementación, el tiempo de cómputo supone tan solo un 8.5 % del consumido en la plataforma CPU+FFTW.

Enfrentando tiempos sobre GPU, nuestra mejor implementación consume un 43 % del requerido por la alternativa más directa, la que trae el propio lenguaje *BrookGPU*, medidos en igualdad de condiciones sobre el mismo equipo, dotado de una 7800 GTX, para las secuencias de 1024×1024 complejos.

Otra de las implementaciones sobre GPU más rápidas en la literatura en el momento de realizar este trabajo de investigación, la debida a Suma-

naweera y Liu, de *Siemens Medical Imaging*, evaluada sobre una NVIDIA Quadro FX, lograba ser, en palabras de los autores, dos veces más rápida que la FFTW, si bien no explicitaban completamente las condiciones en que enfrentaban ambas implementaciones. Nuestra implementación, sin considerar el movimiento de datos de carga y descarga a GPU, es cuatro veces más rápida que la FFTW cuando se ejecuta sobre un motor nv40, seis veces más rápida sobre una G70, y 10 veces más rápida sobre una G80.

Nótese también la rápida evolución entre familias consecutivas del fabricante NVIDIA: los tiempos de ejecución se redujeron del motor nv40 (Abril 2004) al G70 (Junio 2005) en un 32 %, y de éste al G80 (Noviembre 2006), en un 23 %.

Hay dos trabajos especialmente relacionados con nuestra propuesta: el de Jansen *et al.*, que aplica la misma variante, aunque la denominan *split-stream-FFT*, y se ciñen al problema 1D; y la de *Horn* et al.[4], que implementa sobre el mismo lenguaje la misma variante pero decimando en frecuencia en lugar de en tiempo, y sin tomar nunca más de dos nodos por mariposas: la multimiensionalidad la lograban aplicando la propiedad de separabilidad.

A diferencia de en estos trabajos, nuestra propuesta establece un marco teórico donde confluyen el estudio de las distintas variantes de la FFT y el procesado de *streams*, a resultas del cual es posible proponer implementaciones especialmente eficientes al trabajar sobre GPUs que permiten múltiples salidas por *kernel*: aventajando en más del doble de rendimiento, a la mejor alternativa sobre GPU; y hasta 10 veces a la mejor alternativa sobre CPU.

Aún con todo, los tiempos en que se incurre para secuencias de millones de datos, hacen aún inviable su aplicación al reenfoque digital en tiempo real, máxime si consideramos que para pasar de las estudiadas 2D a las 4D que aquel problema requiere aún habría que dar otra pasada, siguiendo la propiedad de separabilidad de la FFT. Por ello, posteriormente abordaremos elestudio del reenfoque fotográfico digital sin pasar por el dominio transformado de Fourier.

Pero la contribución a la FFT sobre GPU no es en vano. No solo las mejoras de tiempo conseguidas son considerables, y en un algoritmo clave en un sinfín de campos de aplicación, sino que además el marco teórico desarrollado es de aplicación a otras arquitecturas no convencionales: véase al respecto las aportaciones del autor en el campo de las FPGA [Rodríguez-Ramos 08], [Montilla 08]; y será de gran utilidad en las nuevas arquitecturas y lenguajes por venir, como los *Intel multi-core* y *Ct*, o el planeado *AMD Fusion* progra-

[4]Esta misma propuesta fue presentada, con cambios mínimos, por un grupo de autores de diversas universidades, entre ellos Horn, ante el Departamento de Defensa de EE.UU. en un estudio sobre la viabilidad de realizar DSP sobre GPUs [Owens 05c].

mado con *Brook+*. Más aún, la rápida evolución de las GPUs juega a nuestro favor, por lo que no es descartable que en un futuro próximo esta misma propuesta algorítmica impulsada por las continuas mejoras del hardware gráfico alcance para trabajar bajo los tiempos límite que nos hemos fijado.

3.11. Aplicación a la Óptica Adaptativa

Los telescopios de base terrestre ven limitado su rendimiento óptico por las alteraciones que la atmósfera ocasiona en los rayos de luz provenientes del espacio. A los procedimientos que contrarrestan estas alteraciones se les conoce como óptica adaptativa y son esenciales para mejorar el aprovechamiento de los telescopios de gran diámetro. Estas alteraciones pueden ser corregidas si se conoce cómo están perturbando la imagen que se obtendría en ausencia de las mismas. Así, si se sabe que se está observando un objeto puntual y sin embargo no se obtiene una imagen acorde del mismo se puede determinar qué tipo de aberraciones está sufriendo la imagen por el hecho de ser observada en esa región del cielo y para ese momento en particular y corregirlas para obtener una imagen tan nítida como el telescopio en uso sea capaz de ofrecer, no solo del objeto conocido, sino de otros más difíciles de caracterizar y que estén situados en la zona de observación caracterizada. Estas alteraciones no son estáticas, por lo que la determinación de las aberraciones del frente de onda y las medidas necesarias para contrarrestarlas tiene un periodo de validez corto. Se considera que el tiempo de estabilidad atmosférica se sitúa alrededor de 10 ms para las observaciones en el rango visible, un poco más en el infrarrojo.

La medida de la fase de frente de ondas es una medida que se lleva a cabo indirectamente. Lo que se sensa es el gradiente de fase de un objeto conocido, por medio de un sensor Shack-Hartmann o análogo. Estas medidas son procesadas hasta conseguir una estimación de la fase del frente de onda que a su vez se transforma en actuaciones que ejercidas, típicamente, sobre un espejo deformable logran restablecer un frente de onda plano, el que corresponde a un objeto situado a una distancia infinita. Los sistemas de óptica adaptativa comprenden pues múltiples pasos pero nos centraremos exclusivamente en el paso de la recuperación de la fase del frente de ondas. Éste es el paso más costoso computacionalmente. Actualmente se usan reconstructores denominados de *multiplicación matriz vector*, VMM, por sus siglas en inglés. Las operaciones requeridas para realizar dicho cálculo por los métodos VMM crece asintóticamente según $O(N^2)$, siendo N el número de actuadores. Y el número de estos se espera que se incremente dramáticamente, desde los cientos hasta las decenas de miles de actuadores, a corto plazo

debido tanto a la implantación de telescopios de diámetros de mayor tamaño así como a la implementación de nuevas aplicaciones de alta resolución sobre los ya existentes.

Se han propuesto diversos métodos para conseguir estimar la fase. Dependiendo de qué se consigue al aplicar los métodos se distingue entre dos tipos de recuperadores. Zonales, aquellos donde lo obtenido es directamente un valor de fase para las distintas zonas, o modales, donde lo que se estiman son los coeficientes de un conjunto de funciones que aproximan la fase en la apertura del telescopio.

Aunque existen variedad de razones para elegir entre unos u otros métodos suelen ser más usados los recuperadores zonales. Son más fácilmente paralelizables lo que permite lograr un rendimiento adecuado con solo aumentar el número de dispositivos de cómputo, ya sean estos DSPs, FPGAs o GPUs. En el pasado ya hemos explorado la posibilidad de implementar algoritmos zonales sobre hardware FPGA y GPU [Marichal-Hernández 05] usando el algoritmo de Hudgin [Hudgin 77], aunque modificado para que funcione para la geometría de un sensor de Shack-Hartmann.

Por su parte, los estimadores modales usan las medidas de los gradientes para ajustar los coeficientes de una expansión de funciones ortogonales sobre la apertura. Dichas funciones suelen ser polinomios de Zernike o exponenciales complejas, aunque se pueden dar otras posibilidades dependiendo de la forma de la pupila. Cuando se usa una pupila anular, lo que es bastante típico en astronomía para acortar el tamaño del telescopio, la característica de ortogonalidad de los polinimios de Zernike se pierde, lo que hace que en ese caso resulte más óptimo optar por funciones de Karhunen-Loève. Para ambos casos, y para otras bases de polinomios complejos, se puede acelerar el cómputo puesto que el kernel de la transformada rápida de Fourier puede describirlos.

Ahora explicaremos cómo, con la ayuda de las FFTs desarrolladas en este capítulo, es posible realizar una implementación sobre GPU de un recuperador modal.

3.11.1. Descripción de un sensor de Shack-Hartmann

Un sensor de frente de onda de Shack-Hartmann muestrea la amplitud compleja del campo electromagnético $\psi_{telescope}(u, v)$ con el fin de obtener un mapa de fases de frente de onda: $\phi(u, v)$. Esto solo es posible si el muestreo se realiza con subpupilas dentro del dominio de coherencia de la fase, r_o. Para ello se introduce un array rígido de microlentes que determinan dicho

ritmo de muestreo. Cada microlente (i,j) produce un punto:

$$I^{ij}(x,y) \;=\; |\; \mathfrak{F}[\psi^{ij}_{telescope}(u,v)] \;|^2$$

Los desplazamientos d_{ij} de los centroides de los puntos con respecto a los centroides de referencia, los que se generarían en una observación ideal, son una estimación proporcional al gradiente de fase media en las subpupilas:

$$d_{ij} \;= K \cdot \frac{\partial \left\langle \phi \right\rangle_{subp_{ij}}}{\partial \vec{r}}$$

Donde $\vec{r}$ es la posición, compuesta por el par (u,v), y K es una constante que depende de la longitud de onda y de las distancias focales del telescopio, la lente reimaginadora y las microlentes. A partir de estas estimaciones de los gradientes de fase se puede recuperar la fase del frente de onda propiamente dicha $\phi(u,v)$ usando una expansión sobre polinomios de exponenciales complejas:

$$\phi(u,v) = \sum_{p,q=0}^{N-1} a_{pq}\; Z_{pq}(u,v) = \sum_{p,q=0}^{N-1} a_{pq}\; \frac{1}{N} e^{\frac{2\pi i}{N}(pu+qv)} = \mathfrak{F}^{-1}(a_{pq})$$

El gradiente en ese caso resulta:

$$\vec{S}(u,v) = \vec{\nabla}\phi(u,v) = \frac{\partial \phi}{\partial u}\vec{i} + \frac{\partial \phi}{\partial v}\vec{j} = \sum_{p,q} a_{pq}\; \vec{\nabla} Z_{pq}$$

Un ajuste mínimo cuadrático del error se obtiene considerando la función objetivo:

$$F = \sum_{u,v=1}^{N} [\vec{S}(u,v) - \sum_{p,q} a_{pq}(\frac{\partial Z_{pq}}{\partial u}\vec{i} + \frac{\partial Z_{pq}}{\partial v}\vec{j})]^2$$

donde $\vec{S}$ son los datos experimentales. Los coeficientes, a_{pq}, de la expansión en polinomios complejos en un reconstructor de fase modal de Fourier se pueden escribir como:

$$a_{pq} \;=\; \frac{i \cdot p \cdot \mathfrak{F}[S^x(u,v)] + i \cdot q \cdot \mathfrak{F}[S^y(u,v)]}{p^2 + q^2} \tag{3.18}$$

Y finalmente la fase se puede recuperar de los datos de gradientes al transformar inversamente estos coeficientes:

$$\phi(u,v) \;=\; \mathfrak{F}^{-1}[a_{pq}] \tag{3.19}$$

Por lo tanto, hay implícitas tres transformadas de Fourier en una recuperación modal de la fase. Debido a la máscara anular de los telescopios la transformada de Fourier introduce grandes frecuencias espaciales que estropean la recuperación de la fase. Pero se puede evitar usando un rellenado distinto de cero y compatible con la definición de gradiente [Poyneer 02] o por medio de iteraciones de Gerchberg [Roddier 91], aunque este último implica realizar cómputos que consumen tiempo en exceso. En cualquier caso el rendimiento que se obtenga va a depender de lo optimizado que esté el cómputo de la FFT sobre el recurso computacional que se emplee.

3.11.2. Recuperación del frente de onda mediante FFT

Nos centraremos en la implementación en GPUs de las ecuaciones (3.18) y (3.19) En la literatura se pueden encontrar otros filtros que reemplazan a la ecuación (3.18) dependiendo del tipo de sensor y la geometría de la pupila. Independientemente de esto, los requisitos computacionales de cualquiera de los posibles filtros son similares, y despreciables en comparación con la carga computacional que suponen las dos transformadas reales directas de la ecuación (3.18), y de la transformada compleja inversa de la ecuación (3.19).

Por lo tanto, nuestra implementación tendrá los siguientes elementos:

1. Simulación de las perturbaciones atmosféricas y del comportamiento del sensor Shack-Hartmann para generar las estimaciones de gradiente S^x y S^y.

2. Expansión de las estimaciones fuera de la máscara anular.

3. Una FFT 2D compleja directa que computa $\mathfrak{F}[S^{xy}]$ donde

$$S^{xy} = S^x + i \cdot S^y.$$

4. Recuperar $\mathfrak{F}[S^x]$ a partir de $\frac{(\mathfrak{F}[S^{xy}]+\mathfrak{F}[S^{xy*}])}{2}$, y $\mathfrak{F}[S^y]$ a partir de $\frac{(\mathfrak{F}[S^{xy}]-\mathfrak{F}[S^{xy*}])}{2 \cdot i}$, donde * es el operador complejo conjugado.

5. El filtrado espacial para computar los coeficientes:

$$a_{pq} = \frac{i \cdot p \cdot \mathfrak{F}[S^x(u,v)] + i \cdot q \cdot \mathfrak{F}[S^y(u,v)]}{p^2 + q^2}$$

6. Una FFT 2D compleja inversa sobre los coeficientes a_{pq}.

La parte de la que se encarga la GPU comienza en el tercer elemento de la lista, mientras que los dos puntos anteriores se computan a priori en la CPU. El punto tercero se puede llevar a cabo directamente con la FFT 2D compleja propuesta en este capítulo. Para acelerar el proceso las dos transformadas reales se ejecutan simultáneamente al trabajar sobre una señal compleja que las contiene a ambas. El cuarto punto es el paso en el que se deshace esa unión, *unfolding*, para recuperar aisladamente los gradientes transformados $\mathfrak{F}[S^x]$ y $\mathfrak{F}[S^y]$. Esta separación y el filtrado se llevan a cabo en GPU haciendo uso de *gather streams*. La transformada inversa se realiza reusando el código que realiza la transformada directa, aplicando la propiedad:

$$\mathfrak{F}^{-1}(X) = \frac{\mathfrak{F}(X^*)^*}{size(X)}.$$

La FFT requiere que los factores de *twiddle* estén precomputados y almacenados como *streams* de solo lectura. Las operaciones de inversión de bit de índice a realizar en el paso de decimación requieren el cómputo a priori de secuencias de inversión que también se hallan precomputadas y almacenadas como *gather streams*. Los pesos del filtro sin embargo se pueden computar en los kernel en GPU pues solo dependen de la posición que está siendo computada por cada *strel*.

En estos cómputos se aplica lo discutido en torno a la subfigura 3.6.**d)** y se hace por tanto necesario el uso de streams auxiliares para realizar el cómputo dando salida a múltiples *streams*.

3.11.3. Resultados y análisis

Este trabajo se llevó a cabo sobre un PC con una CPU AMD XP 3500+ dotado de 1GB de RAM DDR400 sobre una placa madre que empleaba un chipset nForce4. El sistema operativo fue Microsoft Windows XP Professional Edition con SP2. La tarjeta gráfica una nVidia GeForce 7800 GTX con 256 MB de RAM. La tarjeta gráfica se conectó al sistema anfitrión por un bus PCI-Express 16x. La versión del driver nVidia para Windows usada fue la 8198, actualizada en Febrero de 2006.

La GPU se programó usando *BrookGPU* version 0.4, actualizada al CVS en Marzo de 2006. Las simulaciones que produjeron los datos de entrada se programaron con *IDL* y *Octave*.

El cuadro 3.6 contiene los tiempos de ejecución requeridos por los distintos elementos del recuperador de fase. Son tiempos promedios de 1000 ejecuciones. Los tamaños de entrada van desde 64×64 subpupilas de rejilla de muestreo a 1024×1024. Como se observa los cómputos de la FFT consumen la mayor parte del tiempo.

Tarea	Tamaño de entrada				
	64×64	128×128	256×256	512×512	1024×1024
carga de datos a GPU	0’016	0’047	0’234	1’750	7’625
FFT 2D directa	1’750/**1’578**	1’984/**1’688**	2’235/**1’781**	5’891/**4’984**	26’313/**21’781**
Unfolding	0’109	0’109	0’234	0’796	2’922
Filtrado	0’094	0’094	0’188	0’641	2’609
Inversión de bit de índice	0’047	0’047	0’188	1’234	8’484
FFT 2D inversa	1’781/**1’609**	2’000/**1’687**	2’391/**1’938**	7’079/**6’234**	34’578/**30’531**
Volcado de datos desde GPU	0’078	0’203	0’609	2’312	9’110
Tiempo total	3’875/**3’531**	4’484/**3’875**	6’079/**5’172**	19’703/**17’951**	91’461/**82’882**

Cuadro 3.6: Tiempos de ejecución en milisegundos de las distintas partes del recuperador de fase. Los tiempos en **negrilla** en las filas “FFT 2D directa/inversa” y “Tiempo total” son una estimación del tiempo que se obtendría si la extensión MRT fuera capaz de dar salida simultanea a dominios no solapados del mismo *stream*.

Tamaño	Real	Compleja
64×64	50'25 μs	114'50 μs
128×128	313'00 μs	811'00 μs
256×256	1'98 ms	5'48 ms
512×512	9'95 ms	24'17 ms
1024×1024	50'01 ms	113'42 ms

Cuadro 3.7: Tiempos de cómputo de la librería FFTW 3.1.1 sobre una CPU AMD XP 3500+, 2211 GHz, con 512 KB cache L2. La FFTW se ejecutó en modo paciente para secuencias reales y complejas.

Para un problema de tamaño 256×256 los tiempos de ejecución con la implementación propuesta se encuentran cercanos a los 6 ms, lo que supone una gran mejora sobre lo que se obtiene con el uso de librerías FFT altamente optimizadas sobre CPUs de gama alta. También es resaltable que en los problemas de tamaño inferior a 256×256 el rendimiento es bajo debido al costo de configurar y ejecutar los cómputos en la arquitectura de la GPU, lo cual introduce una latencia sistemática por el mayor *pipeline* de éstas.

Los tiempos para ejecutar una única FFT sobre una CPU usando la librería FFTW se muestran en el cuadro 3.7. Se aprecia que incluso sin considerar los tiempos del filtrado es imposible rebajar el tiempo total del recuperador por debajo de los 10, 50 y 220 ms para los tamaños 256×256, 512×512 y 1024×1024 respectivamente, en tanto que el recuperador de fase debe realizar al menos dos FFTs. Sin embargo en los tamaños 64×64 y 128×128 la FFTW ofrece menores tiempos que nuestra implementación en GPU. Un detalle a tener en cuenta es que la librería FFTW tiene la capacidad de trabajar en doble precisión mientras que la GPU no.

Nuestra implementación en su forma actual se ejecuta sobre el entorno de tiempo de ejecución del lenguaje BrookGPU y este a su vez sobre OpenGL. BrookGPU además no está complentamente optimizado. Por ejemplo la ausencia en BrookGPU de una función de copia específica y optimizada entre *streams* debe ser tenida en cuenta al analizar el rendimiento obtenido. Los tiempos en fuente negrilla en el cuadro 3.6 son una estimación de lo que se podría obtener si se pudiera acceder –dejando de lado los límites que impone, pero también la abstracción que brinda BrookGPU–, a las funciones de bajo nivel específicas para copias entre *streams*.

Otro aspecto a considerar son los diferentes ritmos de evolución de las arquitecturas GPU y CPU. Las tendencias actuales en arquitectura de computadores[Owens 05a] indican que la GPU y resto de arquitecturas de *stream* van a ser capaces de mantener el crecimiento de su poder de cómputo

a corto y medio plazo, mientras que las CPUs y el resto de arquitecturas secuenciales verán como este poder de cómputo llega a una meseta, debido al decreciente efecto de mejora sobre las latencias de memoria que producen los avances en miniaturización de circuitos. Por ejemplo en la generación previa de tarjetas gráficas nVidia, una Geforce 6800 ULTRA, los tiempos para un tamaño 512×512 es de 26'60 ms. Esto implica un 25 % de ganancia observada entre dos tarjetas comercializadas con un año de diferencia. Esta ganancia, en el lado de las CPUs, hubiera requerido varias generaciones de procesadores.

Por último, ambas arquitecturas CPU y GPU, pueden colaborar en el mismo computador. Lo que esperamos es que en un futuro cercano la GPU, o su evolución arquitectural, sean los coprocesadores a los que descargar las tareas de cómputo más demandante, mientras que las arquitecturas secuenciales permanecerán como las responsables de la lógica de programa y la entrada/salida. Nuestra propuesta teórica para amoldar la FFT a una arquitectura *SIMD multithreaded* seguramente mantendrá su validez en tal contexto.

Teniendo en cuenta todo lo dicho, es probable que un recuperador para tamaños 512×512, y el búcle completo de AO incluyendo la estimación de centroides para un tamaño 256×256 sean posibles dentro del margen de 5 ms en un futuro cercano con el enfoque colaborativo CPU-GPU.

Capítulo 4

Transformada aproximada del *focal stack*, *aFST*

Comenzaremos con la revisión de los trabajos anteriores sobre la transformada de Radon, los algoritmos para su cómputo y la relación con el reenfoque. Luego revisaremos la formulación de los autores Götz y Druckmüller para el caso 2D, modificándola ligeramente para que resulte posible su extensión al cómputo sobre dominios de 3 y 4 dimensiones. A continuación, reduciremos el número de operaciones de la transformada 4D aDRT, eliminando el cómputo de aquellas integrales innecesarias en el problema del reenfoque fotográfico digital, a resultas de lo cual podremos finalmente formular nuestra *«transformada aproximada del* focal stack*»*, *aFST*, que, gracias a esta rebaja, exhibirá una complejidad computacional lineal con el tamaño del problema. Detallaremos cómo realizar transformadas discretas rápidas de Radon para problemas de tamaño no potencia de dos, y concluiremos mostrando cómo trasladar a algoritmos las transformadas propuestas, que implementaremos tanto para CPU como para GPU.

4.1. Transformada de Radon, algoritmos rápidos y reenfoque

La transformada analítica de Radon, $\mathfrak{R}f(\theta, \rho)$, de una función $f(x, y)$, con $(x, y) \in \mathbb{R}^2$, calcula las integrales de dicha función sobre el espacio de líneas parametrizadas por θ, ángulo de las líneas respecto del eje y, y ρ, distancia al origen [Radon 17]. Esto es,

$$\mathfrak{R}f(\theta, \rho) = \iint f(x, y)\ \delta(x\ cos\ \theta + y\ sin\ \theta - \rho)\ \mathrm{d}x\ \mathrm{d}y, \qquad (4.1)$$

o, alternativamente, usando la forma «pendiente–desplazamiento» con $|s| < 1$,

$$\mathfrak{R}f(s,d) = \int f(u,\ u\,s + d)\ \mathrm{d}u, \tag{4.2}$$

donde s es la pendiente, «*slope*» y d es el desplazamiento, «*displacement*», para las regiones con $|\theta| \leq \frac{\pi}{2}$, o lo que es lo mismo, $|y| \leq |x|$, y

$$\mathfrak{R}f(s,d) = \int f(u\,s + d,\ u)\ \mathrm{d}u, \tag{4.3}$$

para las regiones con $|\theta| \geq \frac{\pi}{2}$.

Cuando se trata de dominios discretos, se usaría la transformada discreta de Radon, *DRT* [Toft 96]. Para computarla existen algoritmos de complejidad cuasi-lineal. Es el caso de los denominados métodos multiescala, en los cuales, las integrales de línea se aproximan por sumatorios de los valores en los nodos —píxeles si hablamos de una imagen— ubicados en la correspondiente banda de ancho unitario, como proponen Brady, en [Brady 92] y [Brady 98]; Götz & Druckmüller en [Götz 96]; y Brandt & Dym en [Brandt 99]; e ilustramos en la figura 4.1.

La DRT también se puede expresar en términos de la transformada de Fourier, [Beylkin 87], en ese caso, se obtiene un algoritmo rápido al usar la FFT como herramienta de cómputo. De este modo, Averbuch *et al.* discuten en [Averbuch 01], [Averbuch 08a] y [Averbuch 08b] cómo computar la DRT pero ahora manteniendo —al contrario que en las discretizaciones propuestas por los otros autores— las propiedades algebraicas de la transformada analítica.

Para referirnos a las transformadas resultantes de los métodos multiescala usaremos el término DRTs *«aproximadas», aDRTs*[1], en tanto que las suposiciones que conducen a dichos algoritmos no cumplen las propiedades de la transformada en el continuo. Siguiendo en la línea del cómputo basado en la FFT pero respetando las propiedades de la transformada analítica, en [Nava 08] se muestra una propuesta de *«transformada rápida discreta del* focal stack*»* de la que soy coautor.

El objetivo de esta sección es obtener una versión apróximada de la FST que cumpla con nuestros requerimientos. Para ello utilizaremos métodos multiescala ya que son suficientemente precisos para las aplicaciones de nuestro interés, además de poder ser computadas rápidamente al no implicar ni multiplicaciones ni operaciones trigonométricas, siendo más rápidas que los basados en FFTs. Diremos también que presentan una transformada inversa

[1]En adelante, daremos por sobreentendidos los calificativos «discreta» y «rápida» en aquellas transformadas que califiquemos como «aproximadas».

rápida y exacta [Press 06], lo cual, aún no siendo crítico para la discusión actual, es una ventaja.

Tomaremos como punto de partida el enfoque multiescala 2D de Götz y Druckmüller, G&D en adelante, y nos basaremos en que el operador de formación fotográfica a partir de un *lightfield*, ecuación (2.3), página 31, puede ser visto como un caso particular de la trasformada 4D de Radon en d-planos, ver sección 2.2.4, cuando se extiende para computar las integrales de planos en un hipercubo de la siguiente manera:

$$\Re f(\boldsymbol{s}, \mathbf{d}) = \int f(\mathbf{u}\,\boldsymbol{s}+\mathbf{d},\ \mathbf{u})\,\mathrm{d}\mathbf{u} = \iint f(u_1\,s_1+d_1,\ u_2\,s_2+d_2,\ u_1,\ u_2)\,\mathrm{d}u_1\,\mathrm{d}u_2 \tag{4.4}$$

No nos consta que se haya profundizado en el problema de la 4D DRT con anterioridad a este trabajo. Sí existen algoritmos para la DRT 3D usando ambos enfoques, multiescala y basados en FFT. En las 3D, la transformada de Radon evalúa el conjunto de integrales a través de los planos que cruzan un volumen. El enfoque multiescala ha sido estudiado por Wu y Brady [Wu 98]; mientras que el basado en Fourier, lo encontramos en Averbuch y Shkolnisky [Averbuch 03]. Ambos trabajos aprovechan la propiedad de separabilidad de las transformadas, que permite computar una transformada multidimensional aplicando repetidamente transformadas sobre un menor número de dimensiones: normalmente N transformadas unidimensionales para computar una transformada N-dimensional.

4.2. Propuesta de modificación de la 2D aDRT de G&D

Consideremos la definición de la DRT aproximada dada por G&D, para una imagen de tamaño $N \times N$, con $N = 2^n, n \in \mathbb{Z}$.

$$\tilde{f}(s,\ d) = \sum_{\mathbf{u}\in\mathbb{Z}_2^n} f(\lambda(\mathbf{u}),\ \ l_s^n(\mathbf{u}) + d), \tag{4.5}$$

donde $\lambda(u_0, \ldots, u_{n-1}) = \sum_{i=0}^{n-1} 2^i u_i$ traduce un vector de coeficientes binarios $\mathbf{u} = (u_0, \ldots, u_{n-1}) \in \mathbb{Z}_2^n$, a su correspondiente entero, $u \in \mathbb{Z}$; y la función $l_s^n : \mathbb{Z}_2^n \to \mathbb{Z}$, definida como,

$$l_s^0(\mathbf{u}) = 0,$$

$$l_s^n(u_0, \ldots, u_{n-1}) = l_{\lfloor s/2 \rfloor}^{n-1}(u_0, \ldots, u_{n-2}) + u_{n-1} \left\lfloor \frac{s+1}{2} \right\rfloor \tag{4.6}$$

determina la discretización de la línea recta $y{=}x\,\frac{s}{N-1}{+}d$, que aproximamos con la banda de un pixel de ancho que atraviesa la imagen a través del conjunto de píxeles con coordenadas $\{\,(0,\ d),\ \ldots\ ,\ (\lambda(\mathbf{u}),\ l^n_s(\mathbf{u})+d),\ \ldots\ ,\ (N{-}1,\ s{+}d)\,\}$.[2]

El sumatorio en $\mathbf{u}$ se puede descomponer en múltiples sumatorios en índices binarios $u_0, \ldots, u_{n-1}$, los cuales se pueden resolver de uno en uno, obteniendo el resultado deseado tras n etapas de transformación parcial:

$$\begin{array}{ccccc} f(x,y) & & \xrightarrow[]{\overset{n}{etapas}} & & \tilde{f}\,(s,d) \\ \Updownarrow & & & & \Updownarrow \\ \tilde{f}^{\,0}(\mathbf{v},d) & \xrightarrow{\overset{m}{etapas}} & \tilde{f}^{\,m}(\mathbf{v},d) & \xrightarrow{\overset{n-m}{etapas}} & \tilde{f}^{\,n}(\mathbf{v},d) \end{array}$$

En las etapas intermedias, considerando m el número de índices ya transformados, la aDRT parcial resulta

$$\tilde{f}^{\,m}(\mathbf{v}|\ d) = \tilde{f}^{\,m}(\underbrace{v_0, \ldots, v_{m-1}}_{\underline{\mathbf{v}}}, \underbrace{v_m, \ldots, v_{n-1}}_{\overline{\mathbf{v}}}\ |\ d) = \sum_{\mathbf{u}\in\mathbb{Z}_2^m} f(\lambda(\mathbf{u},\overline{\mathbf{v}})\ |\ l^m_{\lambda(bit-reversal(\underline{\mathbf{v}}))}(\mathbf{u}){+}d). \tag{4.7}$$

Donde hemos usado el símbolo | para separar los parámetros en f, como en $f(x \mid y)$; y reservamos las comas para separar las dimensiones binarias en las que descomponemos a su vez cada parámetro, como en $f(x_0, x_1 \mid y_0, y_1)$.

Como veremos posteriormente necesitamos distinguir la pendiente de las dimensiones espaciales por lo que proponemos dos modificaciones a la notación de G&D que nos ayudarán en la rebaja del cómputo. Considérense los roles de los m índices binarios menos significativos de $\mathbf{v}$, denotados por $\underline{\mathbf{v}}$, y los $n{-}m$ Más Significativos, $\overline{\mathbf{v}}$. Es conveniente distinguir entre ellos, en lugar de considerar que existe un $\mathbf{v}$ que los engloba, dado que este $\mathbf{v}$ global no es un parámetro que podamos definir geométricamente de manera natural, sino la mezcla de dos que sí lo son: la primera parte, $\underline{\mathbf{v}}$, contiene la información relativa a los bits ya transformados de la pendiente; mientras que $\overline{\mathbf{v}}$ representa a los aún no transformados en la dimensión espacial, en este caso x,

[2] Por mantener la nomenclatura de otros autores no cambiaremos la denominación del parámetro s, *pendiente*, si bien para la definición dada sería más apropiado usar *«ascenso»*. Estrictamente hablando, s es el número de píxeles que se asciende en vertical al avanzar N en horizontal, no la pendiente, que vendría dada por: $\frac{s}{N-1}$. Esta forma de trabajar tiene la particularidad de que el espacio de llegada del algoritmo es regular en *ascensos* medidos sobre el extremo del dominio, no en el ángulo de inclinación de las líneas.

sobre la que se lleva a cabo la integración[3]. Por lo que proponemos trabajar con $\tilde{f}^m$ compuesto por tres parámetros: la *pendiente*, la *franja horizontal* y el *desplazamiento vertical*.

Nótese también que el transformación de índices desde la variable de integración a la pendiente, se da entre posiciones menos significativas en un lado y Más Significativas en el otro. Por lo que proponemos que se reubiquen en las transformadas parciales convenientemente según van apareciendo de manera que nos podamos ahorrar la inversión final de bits que describe [Götz 96].

En resumen, los cambios que introduciremos con respecto a la notación usada en (4.7) son: *a)* denotar la pendiente, antes $\underline{\mathbf{v}}$, por $\boldsymbol{s}$; *b)* distinguir entre el índice binario de la pendiente que se transformará en la etapa m, s_{n-m}, de los ya transformados, $\boldsymbol{\sigma}$; y *c)* denotar los índices Más Significativos de la dimensión de integración, anteriormente $\overline{\mathbf{v}}$, como nuestra $\mathbf{v}$.

La fórmula resultante es

$$\tilde{f}^m(\overbrace{s_{n-m}, \overbrace{s_{n-m+1}, \ldots, s_{n-1}}^{\boldsymbol{\sigma}}}^{\boldsymbol{s}} \mid \overbrace{v_m, \ldots, v_{n-1}}^{\mathbf{v}} \mid d) = \sum_{\mathbf{u}\in\mathbb{Z}_2^m} f(\lambda(\mathbf{u},\mathbf{v}) \mid l^m_{\lambda(\boldsymbol{s})}(\mathbf{u}) + d). \quad (4.8)$$

Esta parametrización de la aDRT 2D parcial, $\tilde{f}^m$, conduce a la siguiente ecuación de mapeo entre dos etapas consecutivas,

$$\tilde{f}^{m+1}(\underbrace{\overbrace{s_{n-m-1}, \overbrace{s_{n-m}, \ldots, s_{n-1}}^{\boldsymbol{\sigma}:\, m\ bits}}^{\boldsymbol{s}:\, m+1\ bits} \mid \overbrace{v_{m+1}, \ldots, v_{n-1}}^{\mathbf{v}:\, n-m-1\ bits}}_{n\ bits} \mid d) = \tilde{f}^m(\boldsymbol{\sigma} \mid 0, \mathbf{v} \mid d) + \tilde{f}^m(\boldsymbol{\sigma} \mid 1, \mathbf{v} \mid d + s_{n-m-1} + \lambda(\boldsymbol{\sigma})) \quad (4.9)$$

según se detalla en [Götz 96, sección IV]. Para proponer la *aFST* desarrollaremos una relación análoga a ésta, de la cual se puede encontrar la derivación en el apéndice B.

4.3. Análisis 2D y extensión a 3D y 4D

Antes de introducir las aDRT para 3D y 4D, analicemos sucintamente el funcionamiento del método multiescala para el caso 2D. Para mayores detalles referirse a [Brady 98].

[3] En el texto nos referiremos por variable de integración a la que ejerce de índice del sumatorio discreto.

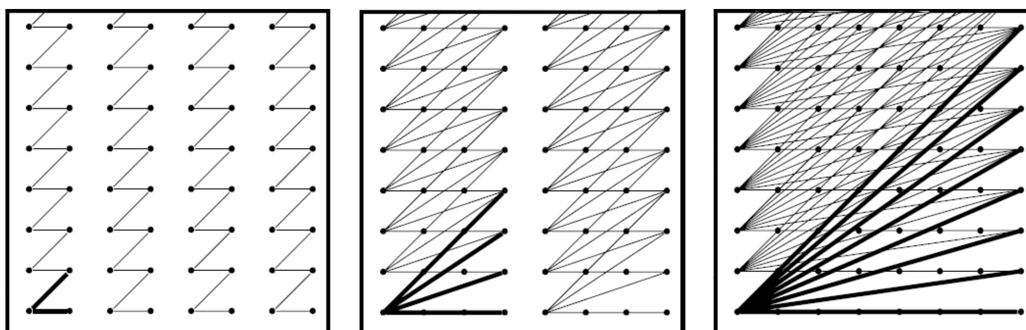


Figura 4.1: Combinación multiescala de segmentos de línea.

En la figura 4.1, tomada de ese trabajo, observamos como es posible evaluar todas las líneas[4] que atraviesan el plano si reducimos inicialmente el problema hasta llevarlo a su expresión mínima, y procedemos luego, poco a poco, a doblar su tamaño. La etapa más básica es aquella que considera el plano dividido en franjas de ancho 2 a lo largo de la variable de integración.

A esa escala, toda «línea discreta» —según la discretización fijada mediante la ec. (4.6)— que forma parte del haz que cruza por un cierto nodo, tiene solo dos posibilidades a su paso por éste. Incrementar su pendiente (recuérdese: *ascenso*), o mantenerla. Esto es, localmente, toda línea que atraviesa el nodo $(2\,x, y)$ incluye a uno de los dos segmentos de línea que ligan dicho punto con $(2\,x{+}1, y)$ y $(2\,x{+}1, y{+}1)$. Estos segmentos son las líneas más simples posibles —no hay segmentos de menos de 2 nodos—, pero contienen ya toda la información necesaria para resolver el problema a mayor escala. No hay combinación de nodos vecinos a escala 2 que no haya sido tenida en cuenta, por lo que es solo cuestión de operar adecuadamente reusando estos segmentos, para dar lugar a todas las combinaciones de «líneas discretas» a través de 4 nodos. Luego, con las de 4 nodos montar las de 8, y así sucesivamente.

Esta idea se plasma formalmente mediante las cuatro ecuaciones que hemos visto para el caso 2D, y son las que debemos extender a 3 y 4 dimensiones:

- La definición analítica, $\Re f(s,\, d)$, ec. (4.2). Que determina cuál será la transformada integral en el continuo.

  Sobre el plano, la transformada de Radon, evalúa integrales de línea. En un dominio de mayor dimensionalidad se abre un abanico de po-

[4] El método que se describe cubre el rango de pendientes $[0°\!: 45°]$. Para considerar 180°, habría que repetir el proceso otras 3 veces modificando adecuadamente la disposición de los datos, o el algoritmo, para que operase sobre ese rango alterado en +45°, +90° y + 135°.

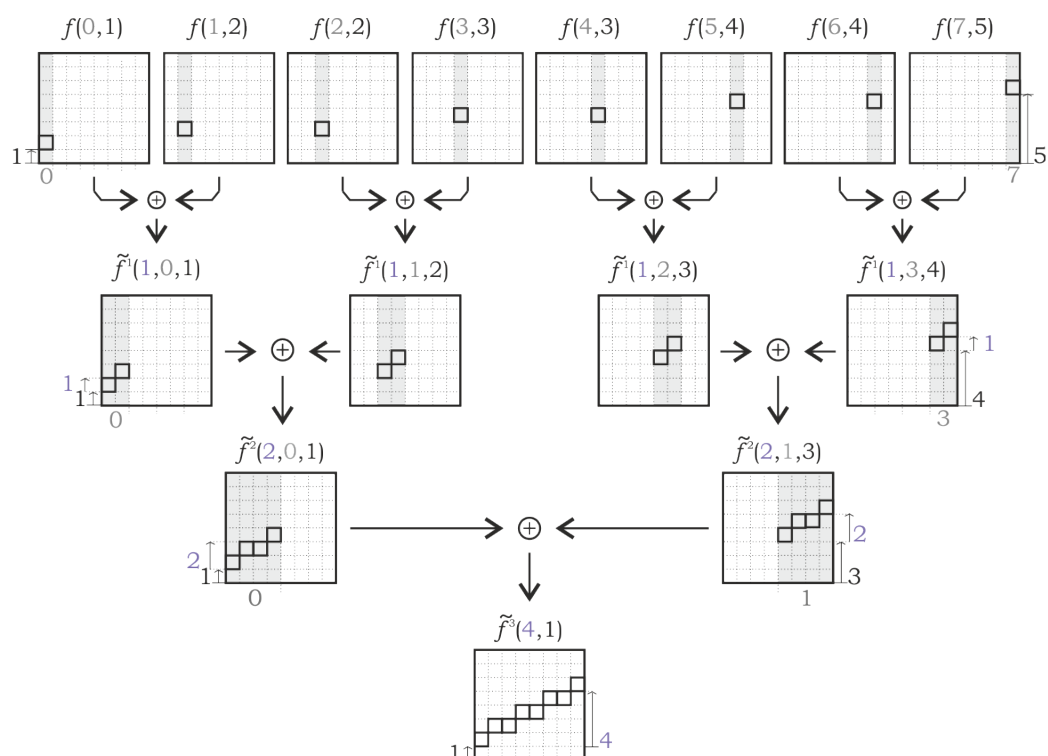


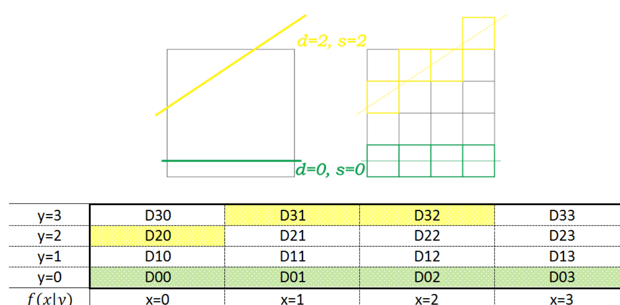
Figura 4.2: Sumas involucradas en el cómputo del valor $\tilde{f}^{\,3}(4, 1)$ sobre una rejilla 8×8.

sibilidades acerca de cuántas y cuáles dimensiones integrar, y cómo se relacionan las variables de integración con las no integradas. Por lo que será importante comenzar definiendo el problema continuo que aproximaremos.

- La transformada aproximada, $\tilde{f}(s, d)$, ec. (4.5); y la transformada parcial, $\tilde{f}^m$, ec. (4.8). Donde un sumatorio sustituye a la integral, y donde las relaciones entre las variables de integración y las no integradas se rigen por la función (4.6). Conllevan la división implícita del problema en «múltiples escalas».

  Los resultados intermedios, los mismos que ilustra la figura 4.1, no son otra cosa que la evaluación parcial de los sumatorios. Así, en el caso 2D, $\tilde{f}^{\,1}$ contiene las sumas sobre los segmentos de ancho 2, $\tilde{f}^{\,2}$ sobre los de ancho 4, etcétera.

- El mapeo entre etapas consecutivas, $\tilde{f}^{\,m} \longrightarrow \tilde{f}^{\,m+1}$, ec. (4.9). Las ecuaciones anteriores finalmente conducen a ésta, que es la clave del algorit-



y=3	D30	D31	D32	D33
y=2	D20	D21	D22	D23
y=1	D10	D11	D12	D13
y=0	D00	D01	D02	D03
$f(x\|y)$	x=0	x=1	x=2	x=3

Figura 4.3: Iustración del cómputo de la 2D aDRT sobre una rejilla 4×4. Arriba a la izquierda) Un dominio 2D con dos líneas contínuas sobre las que deseamos realizar la integración. Arriba a la derecha) La rejilla 4×4 discretizada remarcando los datos que se sumarán para evaluar las integrales de línea propuestas. Nótese que uno de los datos de la línea amarilla cae fuera del dominio inicial. Tabla inferior) Los datos 2D como serán tratados en memoria, accedidos a través de los parámetros x e y. Dentro de cada celda hemos representado los valores discretos, denotándolos como Dxy.

mo, y contiene el modo en que combinar los resultados en cada etapa para dar lugar a los siguientes.

Por ejemplo, en la figura 4.2 se ilustra cuáles son los datos involucrados en el cómputo de un único elemento, $\tilde{f}^{\,3}(4,1)$, sobre una rejilla de tamaño 8×8: qué segmentos de tamaño 4, y, antes, cuáles de tamaño 2, contribuyen a la línea de pendiente 4 que parte de la posición vertical 1. Volviendo sobre los resultados intermedios que quedan definidos en las transformadas parciales, nótese que el número de franjas horizontales se reduce a la mitad a cada incremento de m, pero cada una es el doble de ancha. Igualmente se dobla el máximo ascenso que puede darse en vertical. Esto se debe a la transferencia de índices binarios de una a otra dimensión. Inicialmente, no existirán índices en la pendiente, aparecerán gradualmente, y finalmente, cuando $m{=}n$, solo quedará pendiente, mientras que la información horizontal habrá desaparecido.

Eso si hacemos un análisis *top-down* del algoritmo, pero lo que en realidad sucederá es que se correrá sobre un conjunto de muestras discretas y se calcularán de golpe todas las combinaciones posibles de pendientes y desplazamientos. Pasamos a analizar este enfoque *bottom-up* con la ayuda de las figuras 4.3 y 4.4 en las que se aplica la 2D aDRT a un escenario aún más sencillo: una rejilla 4×4.

En la primera tabla de la figura 4.4 comenzamos por duplicar el tamaño de los datos en su dimensión vertical, d, con la que representaremos el des-

d=7	Z70	Z71	Z72	Z73
d=6	Z60	Z61	Z62	Z63
d=5	Z50	Z51	Z52	Z53
d=4	Z40	Z41	Z42	Z43
d=3	D30	D31	D32	D33
d=2	D20	D21	D22	D23
d=1	D10	D11	D12	D13
d=0	D00	D01	D02	D03
$\tilde{f}^0(v\|d)$	v=0	v=1	v=2	v=3

	s=0	s=1	s=0	s=1
d=7	Z70+Z71	Z70+Z81	Z72+Z73	Z72+Z83
d=6	Z60+Z61	Z60+Z71	Z62+Z63	Z62+Z73
d=5	Z50+Z51	Z50+Z61	Z52+Z53	Z52+Z63
d=4	Z40+Z41	Z40+Z51	Z42+Z43	Z42+Z53
d=3	D30+D31	D30+Z41	D32+D33	D32+Z43
d=2	D20+D21	D20+D31	D22+D23	D22+D33
d=1	D10+D11	D10+D21	D12+D13	D12+D23
d=0	D00+D01	D00+D11	D02+D03	D02+D13
$\tilde{f}^1(s\|v\|d)$	v=0		v=1	

	s=0	s=1	s=2	s=3
d=7	[Z70+Z71] + [Z72+Z73]	[Z70+Z71] + [Z82+Z83]	[Z70+Z81] + [Z82+Z93]	[Z70+Z81] + [Z92+Za3]
d=6	[Z60+Z61] + [Z62+Z63]	[Z60+Z61] + [Z72+Z73]	[Z60+Z71] + [Z72+Z83]	[Z60+Z71] + [Z82+Z93]
d=5	[Z50+Z51] + [Z52+Z53]	[Z50+Z51] + [Z62+Z63]	[Z50+Z61] + [Z62+Z73]	[Z50+Z61] + [Z72+Z83]
d=4	[Z40+Z41] + [Z42+Z43]	[Z40+Z41] + [Z52+Z53]	[Z40+Z51] + [Z52+Z63]	[Z40+Z51] + [Z62+Z73]
d=3	[D30+D31] + [D32+D33]	[D30+D31] + [Z42+Z43]	[D30+Z41] + [Z42+Z53]	[D30+Z41] + [Z52+Z63]
d=2	[D20+D21] + [D22+D23]	[D20+D21] + [D32+D33]	[D20+D31] + [D32+Z43]	[D20+D31] + [Z42+Z53]
d=1	[D10+D11] + [D12+D13]	[D10+D11] + [D22+D23]	[D10+D21] + [D22+D33]	[D10+D21] + [D32+Z43]
d=0	[D00+D01] + [D02+D03]	[D00+D01] + [D12+D13]	[D00+D11] + [D12+D23]	[D00+D11] + [D22+D33]
$\tilde{f}^2(s\|d)$				

Figura 4.4: Sumas realizadas para computar la 2D aDRT sobre una rejilla 4×4. En tres tablas consecutivas se representan los cambios en memoria a medida que se avanza en las transformadas parciales discretas de Radon. Se detalla en el texto.

plazamiento, o punto de corte en vertical. Las celdas 00 a la 33 dan acomodo a los datos originales. Cada celda extra en memoria de rellena con un zero: son los valores Z40 a Z73. La dimensión horizontal la denominaremos v. En estas tablas continuamos con las dos líneas de ejemplo propuestas en la figura anterior, resaltando en amarillo las celdas que participan en el cómputo de $\widetilde{f}^2(2,2)$, y en verde aquellas celdas partícipes en el cómputo de $\widetilde{f}^2(0,0)$.

En la segunda de las tablas, las celdas se indexan por medio de 3 variables: v, s y d. En este paso del algoritmos se sumarán y almacenarán adecuadamente, según marca la ecuación (4.9), cada segmento de línea de dos elementos que surcan la rejilla (ver primer esquema de la figura 4.1). Por ejemplo, la celda $v = 0$, $s = 0$, $d = 0$ almacena la suma de los datos D00 con D01, y representa el sumatorio sobre el segmento de línea que une dos puntos de la rejilla a nivel $v = 0$, la primera de dos subfranjas horizontales, $d = 0$

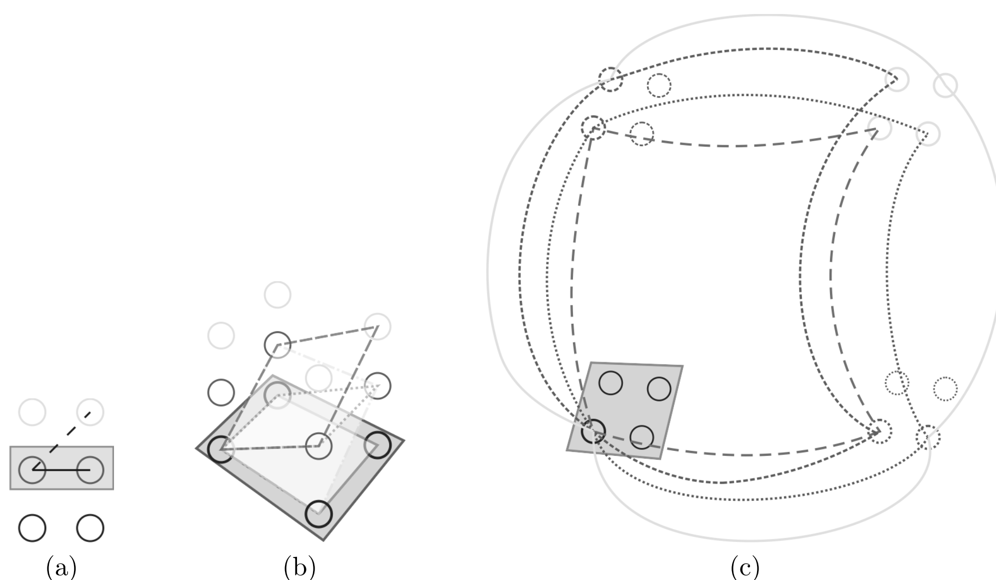


Figura 4.5: Combinaciones entre nodos vecinos en la etapa inicial de la aDRT en 2, 3 y 4 dimensiones.

puesto que corta el eje vertical a esa altura y $s = 0$, sin ascender. Cercana a ella en memoria habrá una celda $v = 0$, $s = 1$, $d = 0$ que representa el mismo concepto, salvo por que en ella se asciende en horizontal 1 elemento, y por lo tanto almacenará la suma de D00 + D11.

En la tercera tabla, los datos son parametrizados nuevamente haciendo uso de dos variables, s y d, puesto que al haber una sola subfranja horizontal la variable v se vuelve innecesaria. Esto coincide con lo que esperábamos, dado que $\widetilde{f}^{\,2}$ constituye la etapa de transformación final y por tanto el resultado para problemas de tamaño 4×4. Las celdas en $s = 0, d = 0$, y $s = 2, d = 2$ almacenan el resultado para las dos líneas de ejemplo propuestas en la anterior figura. Pero los cómputos se han realizado para todas las posibles combinaciones, esto es, barriendo de $s = 0$ a $s = 3$ y de $d = 0$ a $d = 7$, incluso si varias de estas suman contienen valores nulos, al caer las líneas que representan total o parcialmente fuera del dominio inicial. Estos resultados son consecuencia de la aplicación directa de la fórmula (4.9). Al algoritmo se llega simplemente rodeando dicha fórmula de una serie de búcles anidados que evalúan las etapas 0 y 1, y en cada etapa los valores entre 0 y 3 en horizontal (si bien ese rango es compartido entre las variables v y s) y entre 0 y 7 en vertical.

En la subfigura 4.5a, volvemos a presentar las relaciones entre nodos vecinos en la escala básica del caso 2D, sobre la que se asienta la posibilidad

de obtener un algoritmo rápido por reutilización de cómputos entre etapas. En la subfigura 4.5b ilustramos los *patchs* de planos de distintas pendientes a considerar en una vecindad de 4 nodos como etapa básica en la evaluación de todos los posibles planos discretos sobre un volumen 3D. Análogamente, para el caso 4D, en la subfigura 4.5c, y para la definición de transformada de Radon 4D que daremos en (4.15), representamos las 4 posibilidades iniciales que se dan al evaluar ciertos planos discretos sobre un hipercubo.

Con esto en mente, procedemos a definir escuetamente las ecuaciones que rigen las transformadas aproximadas discretas de Radon en 3D y 4D.

4.3.1. Transformada aproximada de Radon en 3D

Las ecuaciones que evalúan (aproximadamente) las integrales a lo largo de los planos descritos por $z - x\ slope_x - y\ slope_y - offset = 0$, en un volumen $f(x, y, z)$, son las siguientes.

En el continuo, definimos la transformada integral

$$\mathfrak{R}f(s_1,\ s_2,\ d) = \int f(u_1,\ u_2,\ u_1\, s_1\ +\ u_2\, s_2\ +\ d)\ \mathrm{d}u_1\, \mathrm{d}u_2, \tag{4.10}$$

cuya aproximación discreta es

$$\tilde{f}\,(s_1|\ s_2|\ d) = \sum_{\mathbf{u_1}\in\mathbb{Z}_2^n} \sum_{\mathbf{u_2}\in\mathbb{Z}_2^n} f(\lambda(\mathbf{u_1})|\ \lambda(\mathbf{u_2})|\ l^n_{s_1}(\mathbf{u_1}) + l^n_{s_2}(\mathbf{u_2}) + d). \tag{4.11}$$

Si definimos la solución hasta la etapa m como

$$\begin{aligned}\tilde{f}^{\,m}(&\overbrace{s_{1_{n-m}}, \ldots, s_{1_{n-1}}}^{\mathbf{s_1}}|\ \overbrace{v_{1_m}, \ldots, v_{1_{n-1}}}^{\mathbf{v_1}}\ | \\ &\qquad\overbrace{s_{2_{n-m}}, \ldots, s_{2_{n-1}}}^{\mathbf{s_2}}|\ \overbrace{v_{2_m}, \ldots, v_{2_{n-1}}}^{\mathbf{v_2}}|\ d) = \\ &\sum_{\mathbf{u_1}\in\mathbb{Z}_2^m} \sum_{\mathbf{u_2}\in\mathbb{Z}_2^m} f(\lambda(\mathbf{u_1}, \mathbf{v_1})\ |\ \lambda(\mathbf{u_2}, \mathbf{v_2})\ | l^m_{\lambda(\mathbf{s_1})}(\mathbf{u_1}) + l^m_{\lambda(\mathbf{s_2})}(\mathbf{u_2}) + d),\end{aligned} \tag{4.12}$$

la solución se obtiene aplicando recursivamente el siguiente mapeo entre etapas.

$$\begin{aligned}\tilde{f}^{\,m+1}(&s_{1_{n-m-1}}, \overbrace{s_{1_{n-m}}, \ldots, s_{1_{n-1}}}^{\boldsymbol{\sigma_1}}|\ \overbrace{v_{1_{m+1}}, \ldots, v_{1_{n-1}}}^{\mathbf{v_1}}| \\ &\qquad s_{2_{n-m-1}}, \overbrace{s_{2_{n-m}}, \ldots, s_{2_{n-1}}}^{\boldsymbol{\sigma_2}}|\ \overbrace{v_{2_{m+1}}, \ldots, v_{2_{n-1}}}^{\mathbf{v_2}}|\ d) = \\ &\qquad\qquad \tilde{f}^{\,m}(\boldsymbol{\sigma_1}\,|\,0, \mathbf{v_1}\,|\,\boldsymbol{\sigma_2}\,|\,0, \mathbf{v_2}\,|\,d) +\end{aligned}$$

$$\widetilde{f}^m(\boldsymbol{\sigma_1}\,|\,1,\mathbf{v_1}\,|\,\boldsymbol{\sigma_2}\,|\,0,\mathbf{v_2})\,|\,d+s_{1_{n-m-1}}+\lambda(\boldsymbol{\sigma_1}))+$$
$$\widetilde{f}^m(\boldsymbol{\sigma_1}\,|\,0,\mathbf{v_1}\,|\,\boldsymbol{\sigma_2}\,|\,1,\mathbf{v_2}\,|\,d+s_{2_{n-m-1}}+\lambda(\boldsymbol{\sigma_2}))+$$
$$\widetilde{f}^m(\boldsymbol{\sigma_1}|1,\mathbf{v_1}|\boldsymbol{\sigma_2}|1,\mathbf{v_2}|\,d+s_{1_{n-m-1}}+\lambda(\boldsymbol{\sigma_1})+s_{2_{n-m-1}}+\lambda(\boldsymbol{\sigma_2})) \quad (4.13)$$

4.3.2. Transformada aproximada de Radon en 4D

Y, equivalentemente, para 4D, definimos,

$$\Re f(s_1,\, s_2,\, d_1,\, d_2) = \int f(u_1,\, u_2,\, u_1\, s_1{+}d_1,\, u_2\, s_2{+}d_2)\, \mathrm{d}u_1\, \mathrm{d}u_2, \qquad (4.14)$$

y su aproximación discreta,

$$\widetilde{f}(s_1\,|\,s_2\,|\,d_1\,|\,d_2) = \sum_{\mathbf{u_1}\in\mathbb{Z}_2^n}\sum_{\mathbf{u_2}\in\mathbb{Z}_2^n} f(\lambda(\mathbf{u_1})\,|\,\lambda(\mathbf{u_2})\,|\,l^n_{s_1}(\mathbf{u_1})+d_1\,|\,l^n_{s_2}(\mathbf{u_2})+d_2). \qquad (4.15)$$

Luego, en la etapa m, la transformada parcial vendrá dada por

$$\widetilde{f}^m(\overbrace{s_{1_{n-m}},\ldots,s_{1_{n-1}}}^{\mathbf{s_1}}\,|\,\overbrace{v_{1_m},\ldots,v_{1_{n-1}}}^{\mathbf{v_1}}\,|\,\overbrace{s_{2_{n-m}},\ldots,s_{2_{n-1}}}^{\mathbf{s_2}}\,|\,\overbrace{v_{2_m},\ldots,v_{2_{n-1}}}^{\mathbf{v_2}}\,|\,d_1|\,d_2) =$$
$$\sum_{\mathbf{u_1}\in\mathbb{Z}_2^m}\sum_{\mathbf{u_2}\in\mathbb{Z}_2^m} f(\lambda(\mathbf{u_1},\mathbf{v_1})|\lambda(\mathbf{u_2},\mathbf{v_2})|l^m_{\lambda(\mathbf{s_1})}(\mathbf{u_1})+d_1|l^m_{\lambda(\mathbf{s_2})}(\mathbf{u_2})+d_2), \quad (4.16)$$

lo cual conduce al siguiente mapeo.

$$\widetilde{f}^{m+1}(s_{1_{n-m-1}},\overbrace{s_{1_{n-m}},\ldots,s_{1_{n-1}}}^{\boldsymbol{\sigma_1}}\,|\,\overbrace{v_{1_{m+1}},\ldots,v_{1_{n-1}}}^{\mathbf{v_1}}\,|\,s_{2_{n-m-1}},\overbrace{s_{2_{n-m}},\ldots,s_{2_{n-1}}}^{\boldsymbol{\sigma_2}}\,|\,\overbrace{v_{2_{m+1}},\ldots,v_{2_{n-1}}}^{\mathbf{v_2}}\,|\,d_1|\,d_2) =$$
$$\widetilde{f}^m(\boldsymbol{\sigma_1}\,|\,0,\mathbf{v_1}\,|\,\boldsymbol{\sigma_2}\,|\,0,\mathbf{v_2}\,|\,d_1\,|\,d_2)+$$
$$\widetilde{f}^m(\boldsymbol{\sigma_1}\,|\,1,\mathbf{v_1}\,|\,\boldsymbol{\sigma_2}\,|\,0,\mathbf{v_2}\,|\,d_1+s_{1_{n-m-1}}+\lambda(\boldsymbol{\sigma_1})\,|\,d_2)+$$
$$\widetilde{f}^m(\boldsymbol{\sigma_1}\,|\,0,\mathbf{v_1}\,|\,\boldsymbol{\sigma_2}\,|\,1,\mathbf{v_2}\,|\,d_1\,|\,d_2+s_{2_{n-m-1}}+\lambda(\boldsymbol{\sigma_2}))+$$
$$\widetilde{f}^m(\boldsymbol{\sigma_1}|1,\mathbf{v_1}|\boldsymbol{\sigma_2}|1,\mathbf{v_2}|d_1+s_{1_{n-m-1}}+\lambda(\boldsymbol{\sigma_1})\,|\,d_2+s_{2_{n-m-1}}+\lambda(\boldsymbol{\sigma_2})) \quad (4.17)$$

4.3.3. Complejidad computacional de las aDRTs

Las ecuaciones (4.9), (4.13) y (4.17), pueden ser trasladadas directamente a algoritmos que computan las transformadas aproximadas discretas de

Radon en 2D, 3D y 4D, tal como las hemos definido. Todos estos algoritmos exhibirán una complejidad cuasilineal, $O(N^b\, log_2\, N)$, aplicados a problemas de tamaño N^b en b dimensiones. Puesto que requerirán $log_2\, N$ etapas, y, en cada etapa, se han de realizar un número de operaciones que está en el orden del tamaño del problema, N^b.

En la primera etapa, es inmediato que para un problema de tamaño N^b habrá N^b elementos a considerar. Pero, ¿cuántos segmentos de dos líneas, sobre el plano, o *patches* de 4 nodos, en el cubo y en el hipercubo 4D, participarán en la segunda etapa? También N^b. Obsérvese gráficamente, en las subfiguras en 4.5, que el dominio de cómputo lo podemos considerar dividido en agrupaciones lógicas —los recuadros de fondo gris envolviendo a los nodos— y que éstos contienen a tantos nodos como posibilidades hay que evaluar para esa dimensionalidad del problema: en 2D se agrupan nodos de dos en dos, y para cada grupo se evalúan 2 segmentos; serán 4 las posibilidades a evaluar en 3D y 4D, sobre vecindades de 4 nodos. Aunque desde un grupo se hagan consultas a nodos pertenecientes a otros grupos, esto se compensa por las consultas, en igual número, que el primero recibe del exterior.

Sobre la fórmula esto es mucho más sencillo de observar: basta con fijarnos en que en las relaciones $\tilde{f}^{\,m} \longrightarrow \tilde{f}^{\,m+1}$ son transformados índices binarios en índices binarios, que pasan de unas dimensiones a otras, pero el tamaño del problema en su conjunto permanece constante entre etapas. Empieza, comienza y termina siendo de tamaño N^b.

4.4. Reenfocando digitalmente con una 4D *aDRT*

Si comparamos el operador fotográfico, ec. (2.3), con la recién definida 4D aDRT, ecuaciones (4.14) y (4.15), observamos que al aplicarse ambas sobre una entrada 4D, el resultado de la primera constituye un subconjunto del resultado que se obtiene por medio de la segunda: el espacio 3D que satisface $s_1 = s_2$. Tal como la hemos definido la 4D aDRT no es óptima en la resolución del problema de reenfoque fotográfico digital, puesto que realiza cómputos sobre planos que no están contenidos en el *focal stack*.

Para eliminar estos cómputos innecesarios —cuando $s1 \neq s2$, los cuales son la mayoría— podríamos recurrir a aplicar técnicas de *pruning* [Markel 71]. Pero los métodos de *pruning* en la literatura que tratan con múltiples dimensiones [Knudsen 93] o regiones dispersas [Hu 05], son más complicadas y menos eficientes que lo que podemos obtener. Haremos uso del aislamiento de las pendientes respecto de los parámetros espaciales, para

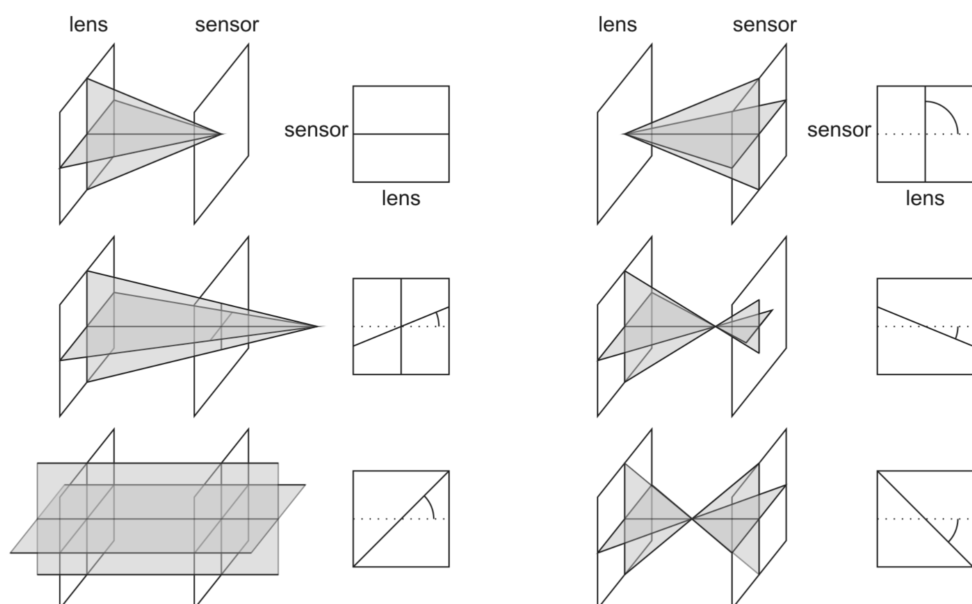


Figura 4.6: Efectos sobre el enfoque al evaluar hiperplanos de distintas pendientes sobre un *lightfield* 4D. Se detalla en el texto.

así podar de manera efectiva los cómputos innecesarios.

De modo que la ecuación continua sería,

$$\mathfrak{R}f(s,\, d_1,\, d_2) = \int f(u_1,\, u_2,\, u_1\; s{+}d_1,\, u_2\; s{+}d_2)\, \mathrm{d}u_1\, \mathrm{d}u_2. \tag{4.18}$$

Para resolverla proponemos la *«transformada aproximada del* focal stack», que computa

$$\tilde{f}\,(s \mid d_1 \mid d_2) = \sum_{\mathbf{u_1} \in \mathbb{Z}_2^n} \sum_{\mathbf{u_2} \in \mathbb{Z}_2^n} f(\lambda(\mathbf{u_1}) \mid \lambda(\mathbf{u_2}) \mid l_s^n(\mathbf{u_1}) + d_1 \mid l_s^n(\mathbf{u_2}) + d_2). \tag{4.19}$$

La figura 4.6 ilustra el hecho de que, aplicado sobre un *lightfield* 4D, la transformada recién definida calcula el *focal stack* 3D aproximado de la escena. Existen 4 dimensiones: que corresponden a recorrer vertical y horizontalmente la lente y el sensor. El efecto de reenfoque se puede ver como una integración donde ambos pares de dimensiones en vertical y horizontal varían simultáneamente: la dimensión horizontal sobre la lente varía con respecto a la dimensión horizontal en el sensor, al mismo ritmo que la vertical sobre la lente lo hace sobre la vertical del sensor.

Existe una única pendiente que relaciona ambos grupos de dimensiones, y será proporcional, sobre el *focal stack* 3D resultante, a la profundidad a

lo largo del eje óptico, z. El papel de las dimensiones horizontal y vertical sobre el *focal stack* lo asumirán los desplazamientos horizontal y vertical en el operador:

$$_{lightfield}(x_{lente}|y_{lente}|x_{sensor}|y_{sensor}) \stackrel{aFST}{\longrightarrow} {}_{focalstack}(z|x|y)$$

Los planos de enfoque que ocupan el espacio imagen que en la dirección del sensor se extiende desde éste hacia el infinito —columna izquierda en la figura 4.6— se logran integrando sobre los hiperplanos definidos por

$$x_{sensor} = pendiente_z \cdot x_{lente} + desplazamiento_x,$$

$$y_{sensor} = pendiente_z \cdot y_{lente} + desplazamiento_y,$$

cuando la $pendiente_z \in [0, 1]$; y los planos que van del punto medio interior del espacio lente–sensor hasta este último —columna derecha salvo diagrama superior— se obtendrán por la integración a lo largo de

$$\{x, y\}_{sensor} = pendiente_z \cdot \{x, y\}_{lente} + desplazamiento_{\{x,y\}},$$

con $pendiente_z \in [-1, 0]$.

Existen otros dos cuadrantes que podrían ser computados, pero son irrelevantes para nuestros propósitos, en tanto que ocupan el espacio entre el punto medio interior hacia la lente y más allá de ésta hasta el infinito.

4.5. Transformada aproximada de *focal stack*

Para la etapa m, definimos la transformada parcial como

$$\begin{aligned}
&\tilde{f}^m(\boldsymbol{s} \,|\, \mathbf{v_1} \,|\, \mathbf{v_2} \,|\, d_1 \,|\, d_2) = \\
&\tilde{f}^m(\overbrace{\overbrace{s_{n-m}, \ldots, s_{n-1}}^{\boldsymbol{s}:\, m\ bits} \,|\, \overbrace{v_{1_m}, \ldots, v_{1_{n-1}}}^{\mathbf{v_1}:\, n-m\ bits} \,|\, \overbrace{v_{2_m}, \ldots, v_{2_{n-1}}}^{\mathbf{v_2}:\, n-m\ bits}}^{2n-m\ bits} \,|\, d_1 | \, d_2) = \\
&\sum_{\mathbf{u_1} \in \mathbb{Z}_2^m} \sum_{\mathbf{u_2} \in \mathbb{Z}_2^m} f(\lambda(\underbrace{\mathbf{u_1},\, \mathbf{v_1}}_{n}) |\, \lambda(\underbrace{\mathbf{u_2},\, \mathbf{v_2}}_{n}) |\, l^m_{\lambda(\boldsymbol{s})}(\mathbf{u_1}) + d_1 |\, l^m_{\lambda(\boldsymbol{s})}(\mathbf{u_2}) + d_2).
\end{aligned} \tag{4.20}$$

Nótese que se trata de una función de dimensión 5, y que el tamaño de 3 de estas dimensiones, $\boldsymbol{s}$, $\mathbf{v_1}$ y $\mathbf{v_2}$, globalmente consideradas, se reduce según avanzan las etapas, lo cual no era el caso para ninguna de las aDRTs previamente definidas. Aparte de la reducción, se sigue cumpliendo el trasvase de índices entre dimensiones. Así, cuando $m = 0$, la dimensión de pendiente,

$\boldsymbol{s}$, estará vacía, mientras que las dimensiones espaciales, $\mathbf{v_1}$ y $\mathbf{v_2}$, lo estarán al final del proceso:

$$\tilde{f}^{\,0}(\emptyset|\underbrace{\mathbf{v_1}|\mathbf{v_2}}_{2n\,bits}|d_1|d_2) \overset{m}{\underset{}{\overset{etapas}{\longrightarrow}}} \tilde{f}^{\,m}(\underbrace{\boldsymbol{s}|\mathbf{v_1}|\mathbf{v_2}}_{2n-m\,bits}|d_1|d_2)\ \ldots$$

$$\ldots\ \tilde{f}^{\,m}(\boldsymbol{s}|\mathbf{v_1}|\mathbf{v_2}|d_1|d_2) \overset{n-m}{\underset{}{\overset{etapas}{\longrightarrow}}} \tilde{f}^{\,n}(\underbrace{\boldsymbol{s}}_{n\,bits}|\emptyset|\emptyset|d_1|d_2)$$

Al comenzar la transformación, $\tilde{f}^{\,0}$ acomodará a los datos 4D de entrada, que serán parametrizados usando 4 dimensiones (puesto que 1 de las dimensiones estará reservada pero aún permanece vacia); a partir de ahí se transforma usando la parametrización 5 dimensional; y finalmente el resultado se maneja como si fuera 3 dimensional (puesto que 2 de las dimensiones se habrán vaciado). Esto no supondrá un incremento en la complejidad del algoritmo, pero sí requerirá una cuidadosa estrategia de direccionamiento de memoria.

La definición dada conduce al siguiente mapeo entre etapas consecutivas de la transformada parcial. Se deriva en el apéndice B.

$$\begin{aligned}
\tilde{f}^{\,m+1}(\,s_{n-m-1},\,\overbrace{s_{n-m},\ldots,s_{n-1}}^{\boldsymbol{\sigma}}\,|\,\overbrace{v_{1_{m+1}},\ldots,v_{1_{n-1}}}^{\mathbf{v_1}}\,|\,\overbrace{v_{2_{m+1}},\ldots,v_{2_{n-1}}}^{\mathbf{v_2}}\,|\,d_1|\,d_2) = \\
\tilde{f}^{\,m}(\boldsymbol{\sigma}|\,0,\mathbf{v_1}|\,0,\mathbf{v_2}|\,d_1|\,d_2)+ \\
\tilde{f}^{\,m}(\boldsymbol{\sigma}|\,1,\mathbf{v_1}|\,0,\mathbf{v_2}|\,d_1+s_{n-m-1}+\lambda(\boldsymbol{\sigma})|\,d_2)+ \\
\tilde{f}^{\,m}(\boldsymbol{\sigma}|\,0,\mathbf{v_1}|\,1,\mathbf{v_2}|\,d_1|\,d_2+s_{n-m-1}+\lambda(\boldsymbol{\sigma}))+ \\
\tilde{f}^{\,m}(\boldsymbol{\sigma}|\,1,\mathbf{v_1}|\,1,\mathbf{v_2}|\,d_1+s_{n-m-1}+\lambda(\boldsymbol{\sigma})\,|\,d_2+s_{n-m-1}+\lambda(\boldsymbol{\sigma}))
\end{aligned} \quad (4.21)$$

La reducción en el tamaño de los datos se puede apreciar más fácilmente en esta fórmula. Para una combinación dada de valores $\boldsymbol{\sigma}$, $\mathbf{v_1}$, $\mathbf{v_2}$, d_1, y d_2, se relacionan dos valores, $s_{n-m-1}=\{0,1\}$, en $\tilde{f}^{\,m+1}$, con cuatro valores de $\tilde{f}^{\,m}$. Así para que apareciera un nuevo índice binario ya transformado en la pendiente, s_{n-m-1}, se han colapsado dos, los Más Significativos, índices binarios de $\mathbf{v_1}$ y $\mathbf{v_2}$ en $\tilde{f}^{\,m}$. Tras cada etapa se requiere globalmente un bit menos para direccionar los datos, lo cual indica que su tamaño se ha dividido por dos. Tras n etapas, habrá desaparecido una dimensión completa.

De este modo, hemos conseguido podar todos aquellos cómputos que se requerían anteriormente, ec. (4.17), para lidiar con dos pendientes distintas.

De no haberla designado en función del problema al que está orientada, como *aFST*, una denominación alternativa hubiera sido *«4D:3D aDRT»*, por su peculiar reducción de dimensionalidad a lo largo de las etapas.

4.6. Algoritmo de la *aFST*

Hemos considerado como multidimensionales los datos que participan en cada una de las transformadas propuestas. Al trasladar las ecuaciones resultantes a algoritmos, los almacenaremos y direccionaremos como *arrays* unidimensionales. Por ejemplo, el dato en $\tilde{f}^m(\boldsymbol{\sigma} \,|\, 1, \mathbf{v_1} \,|\, 1, \mathbf{v_2} \,|\, d_1 + s_{n-m-1} + \lambda(\boldsymbol{\sigma}) \,|\, d_2 + s_{n-m-1} + \lambda(\boldsymbol{\sigma}))$ se almacenará en `fm[ sigma + (1 << m)+ (v1 << m+1)+ (1 << m+n-m)+ (v2 << m+n-m+1)+ (d1+snm1+sigma << m+n-m+n-m)+ (d2+snm1+sigma << m+n-m+n-m+n+1)]`. Para componer esa dirección de memoria, cada parámetro ha sido multiplicado por el tamaño de los que le preceden. Nótese cómo estos tamaños dependen de la etapa en curso, `m`.

El siguiente pseudo-código considera los datos de entrada procedentes de una cámara plenóptica parametrizada en dos planos, con el siguiente orden en memoria `f[xlens, ylens, xsensor, ysensor]`, donde `xlens` es el parámetro de menor significancia. Por simplicidad consideramos el tamaño $N \times N \times N \times N$, con $N = 2^n$. Las variables `xsensor` y `ysensor`, que corresponden a d_1 y d_2, deben extenderse para terminar ocupando $2 \times N$ tal como ocurre en [Götz 96].

Haremos uso de dos arrays: `fmp1` de tamaño `1 << n+n+n+1+n+1`, y `fmp1`, de tamaño mitad.

1. Cópiese los datos de entrada desde `f` a `fm` teniendo en cuenta que el tercer y cuarto parámetro en `fm` han de ocupar el doble de tamaño rellenando con ceros.

2. Aplíquese **`for`** `m := 0` a `n-1`

   - Compútese cada dato en `fmp1` a partir de cuatro datos en `fm`, tal como indica la ec. (4.21), y usando el direccionamiento que se describió anteriomente. Esto implica la ejecución de múltiples búcles anidados: **`for`** `d1 := 0` a `1 << n+1`, **`for`** `d2 := 0` a `1 << n+1`, **`for`** `v1 := 0` a `1 << n-m-1`, **`for`** `v2 := 0` a `1 << n-m-1`, **`for`** `sigma = 0` a `1 << m`, y **`for`** `snm1 := 0, 1`. Téngase también en cuenta que los desplazamientos en los parámetros `d1` y `d2` deben comprobarse para que no se consulten más allá de sus dominios; esto es, `(d1+snm1+sigma)` y similares, no deben rebasar `2*N`.

- Intercámbiese los punteros `fm` y `fmp1`.

3. El resultado en `fm` tras la última etapa (e intercambio), constituye el resultado deseado `focalStack[z, x, y]`, de tamaño $N \times 2N \times 2N$.

4. Un procedimiento análogo se puede realizar para conseguir la mitad del *focal stack* correspondiente a las pendientes negativas. Los desplazamientos en los parámetros `d1` y `d2` pasan a ser negativos, v. g. `(d1-snm1-sigma)`. El tamaño de esas dimensiones deben doblarse al inicio, como antes, pero los datos de entrada en lugar de rellenar la mitad superior del espacio reservado, ocupan la mitad inferior, siendo la superior la que se rellene con ceros. Se introducen sentencias condicionales para garantizar que estos desplazamientos negativos se mantengan por encima de 0.

4.6.1. Complejidad computacional

El algoritmo que acabamos de exponer, realiza n pasadas sobre los datos iniciales, y en cada pasada el tamaño de los datos se reduce a la mitad. En cada una de estas etapas hay que realizar 4 consultas a datos y sumas de los mismos, por cada dato de la nueva secuencia `fmp1`; o 2 por cada dato, en términos del tamaño de `fm`. En total:

$$2N^4 + \frac{2N^4}{2} + \ldots + \frac{2N^4}{2^{n-1}} \;=\; 2N^4\Big(1 + \sum_{i=1}^{n-1} \frac{1}{2^i}\Big) \leq 4N^4.$$

Esto es, una complejidad lineal con el tamaño del problema: $O(N^4)$.

4.6.2. Tamaños no potencias de 2

Las transformadas que hemos definido, ecuaciones (4.13), (4.17) y (4.21), y la derivación de esta última en el apéndice B, asumen que el tamaño de los datos de entrada es $N \times \ldots \times N$, y que N es un número potencia de dos.

Cuando N no es potencia de dos, aún es posible diseñar una transformada aproximada. Pero se requerirá una definición más general de *líneas discretas*, que suplante el papel de la ec. (4.6), en la cual se basan las transformadas multiescala.

Asumamos que el tamaño de datos será $N = b_0 \times b_1 \times \ldots \times b_{n-1}$, y que no necesariamente $b_i = 2,\ \forall i$, tal como era el caso anteriormente: $N = \prod_{i=0}^{n-1} 2 = 2^n$. Si esto es así, un número $u \in [0, N-1]$ se puede descomponer en coeficientes $\mathbf{u} = (u_0, \ldots, u_{n-1}),\ u_i \in [0, b_i]$ tales que $\lambda(\mathbf{u}) = \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} b_j u_i$.

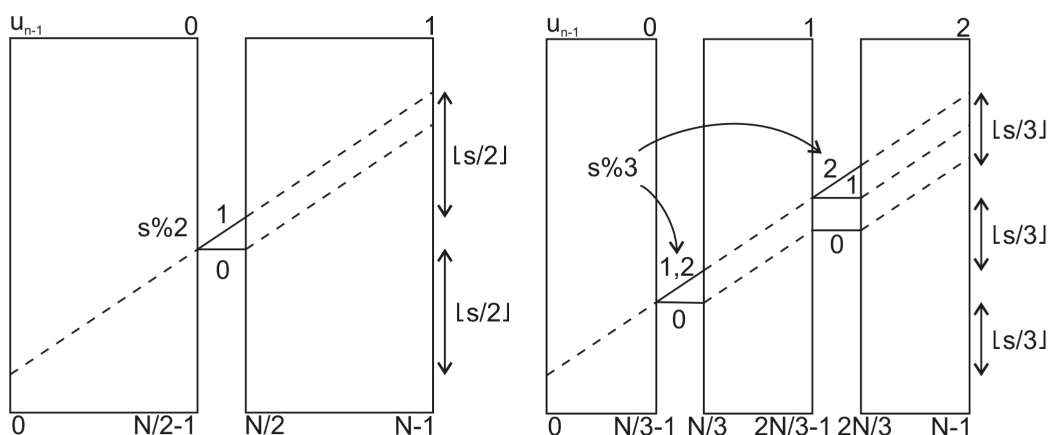


Figura 4.7: Definición recursiva de líneas discretas por medio de una división a la mitad del dominio y por una división en terceras partes.

Adicionalmente podemos definir un operador de *desplazamiento a derecha*, $u \gg i$, como $\lfloor u/(\prod_{j=0}^{i-1} b_j)\rfloor$, y con ese operador calcular los coeficientes u_i como $u_i = (u \gg i)$ % b_i, donde % denota el operador resto de la división entera.

Con todo esto en mente, podemos volver sobre la ecuación (4.6) y observar que es la expresión matemática de la siguiente regla, tal como se ilustra en la figura 4.7:

> si consideramos un dominio dividido en dos mitades de idéntico tamaño, nuestra línea discreta es aquella que cuando alcanza la mitad del dominio, ha llevado a cabo la mitad del ascenso previsto; acerca de cómo actuar dentro de estas dos mitades, aplíquese recursivamente esta regla.

Así, en la ec. (4.6) el primer sumando, $l_{\lfloor s/2\rfloor}^{n-1}(u_0, \ldots, u_{n-2})$, da cuenta de «dentro de estas dos mitades, aplíquese la misma regla», mientras que el segundo, juega el papel de «llevar a cabo la mitad del ascenso», $\lfloor(s+1)/2\rfloor$, «cuando se alcanza la mitad del dominio», u_{n-1}. Nótese que $\lfloor(s+1)/2\rfloor = \lfloor s/2\rfloor + s_0 = (s \gg 1) + s_0$, y que el término $+s_0$, refleja la posibilidad de que s sea un número impar, en cuyo caso este «resto» debe ser «ascendido» en la frontera entre ambas mitades. Obsérvese también, que esta fórmula relaciona un índice binario en la variable de integración con uno en la pendiente/ascenso. Y que la significancia de los índices relacionados en u y s son opuestas: $u_{n-1-m} \Longleftrightarrow s_m$.

Por todo ello, podemos redefinir la ec. (4.6) de tal forma que siga expresando la misma idea que se plasmó como «regla escrita», pero sin asumir que

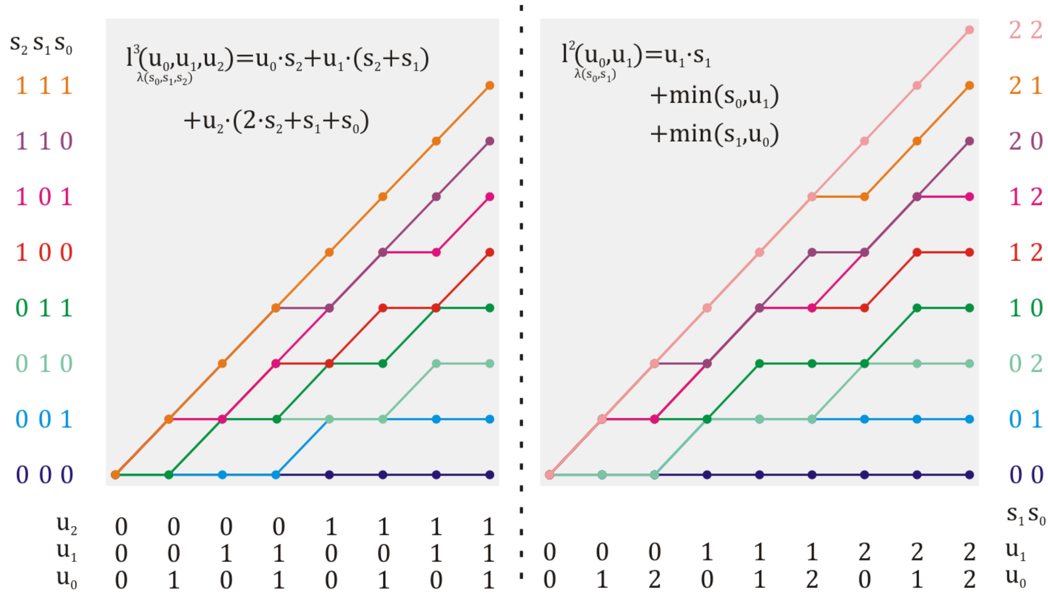


Figura 4.8: Líneas discretas para N=8 y base binaria; y N=9 con base ternaria.

s_m y u_{n-1-m} han de ser necesariamente índices binarios.

$$l_s^n(u_0,\ldots,u_{n-1}) = l_{s\gg 1}^{n-1}(u_0,\ldots,u_{n-2}) + \min(u_{n-1},s_0) + (s \gg 1)u_{n-1} \quad (4.22)$$

Como era de esperar, la ec. (4.6) resulta ser la forma simplificada de (4.22) para el caso $N = 2^n$. El resto del ascenso, s_0, que previamente se realizaba en el único cambio entre subdominios, u_{n-1} y se expresaba por la multiplicación de ambos índices binarios, ahora se distribuirá haciendo uso de $\min(u_{n-1}, s_0)$.

Trasladar los efectos de esta redefinición a las ecuaciones de mapeo entre etapas es directo. Por ejemplo, para el caso 4D:3D, la ec. (4.21) se convierte en:

$$\begin{aligned}
\tilde{f}^{m+1}(&\overbrace{s_{n-m-1}}^{s_{n-m-1}\in\mathbb{Z}_{b_m}}, \overbrace{s_{n-m},\ldots,s_{n-1}}^{\boldsymbol{\sigma}} | \\
&\overbrace{v_{1_{m+1}},\ldots,v_{1_{n-1}}}^{\mathbf{v_1}} | \overbrace{v_{2_{m+1}},\ldots,v_{2_{n-1}}}^{\mathbf{v_2}} | d_1 | d_2) = \\
\sum_{v_{1_m}=0}^{b_m-1} \sum_{v_{2_m}=0}^{b_m-1} &\tilde{f}^m(\boldsymbol{\sigma} | v_{1_m}, \mathbf{v_1} | v_{2_m}, \mathbf{v_2} | d_1 + \min(v_{1_m}, s_{n-m-1}) + v_{1_m}\lambda(\boldsymbol{\sigma}) | \\
&d_2 + \min(v_{2_m}, s_{n-m-1}) + v_{2_m}\lambda(\boldsymbol{\sigma})) \quad (4.23)
\end{aligned}$$

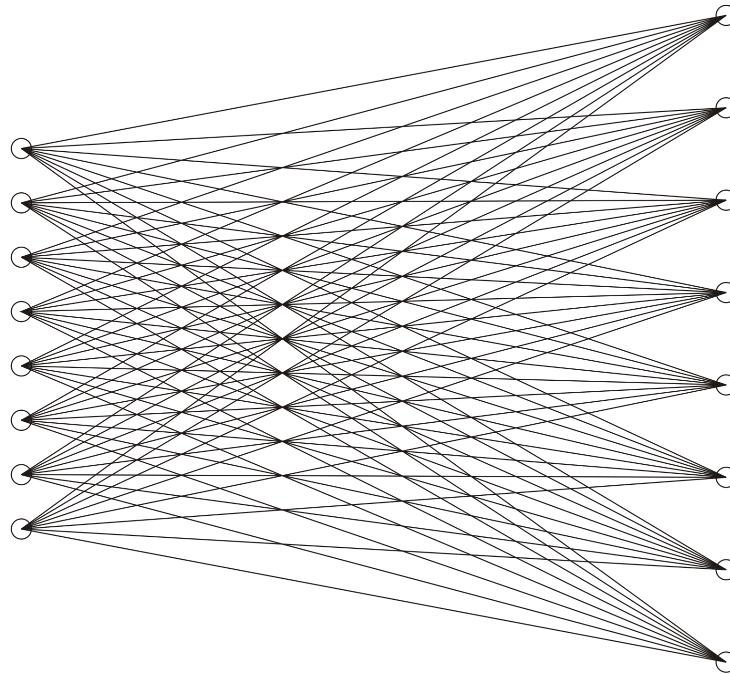


Figura 4.9: Distribución de rayos en el interior de una plenóptica *flatland* con 8 microlentes y 8 píxeles tras microlente. Se detalla en el texto.

4.7. Interpretación óptica de la *aFST*

Esta sección tratará de responder a la pregunta de qué efecto tiene la discretización a la que obliga el empleo de transformadas aproximadas multiescala sobre el problema físico de formación de imagen en una cámara plenóptica. Veámoslo usando una serie de ilustraciones, especialmente las figuras 4.9, 4.13 y 4.15.

4.7.1. Reordenando la imagen plenóptica

La primera de estas figuras representa una idealización, en *flatland*, del efecto que causan las microlentes sobre la formación de imagen en cada pixel del sensor. En el esquema cada línea representa el rayo director de la luz que incide en un pixel. Los nodos circulares representan a un lado, el más estrecho, a cada una de las $M = 8$ microlentes, que hemos considerado cubren el sensor, y al otro, más ancho, a cada una de las $N = 8$ subaperturas en que se divide la apertura de la lente principal por el hecho de colocar 8 píxeles tras cada microlente.

La figura sirve indistintamente para representar la separación de los rayos

en el interior o en el exterior de la cámara oscura de una plenóptica *flatland*. Puesto que el esquema se trasladaría idénticamente fuera de la cámara, donde los nodos de microlentes se exteriorizan aplicando la ley de lentes finas hasta un plano z en espacio objeto conjugado del sensor sobre el espacio imagen. Esta dualidad da pie a dos interpretaciones de la imagen plenóptica con solo reordenar las muestras[5]. Tras ser capturada, sin más procesamiento, la función plenóptica 4D sobre un sensor 2D (función 2D sobre un sensor 1D en *flatland*) sigue el ordenamiento marcado por los píxeles como dimensiones menos significativas y microlentes como Más Significativas, que puede ser interpretada, según lo que acabamos de ver, como un conjunto de $N \times N$ *BRDFs*, cada una de tamaño $M \times M$ muestras, que abarcan el FOV a la distancia del plano de enfoque de una cámara convencional. O si se sigue el orden de píxeles como Más Significativos y microlentes como menos significativas, como una sucesión de $M \times M$ imágenes de $N \times N$ píxeles, cada una con un punto de vista ligeramente distinto, cubriendo la apertura de la lente principal. Ver figuras 4.10 y 4.11.

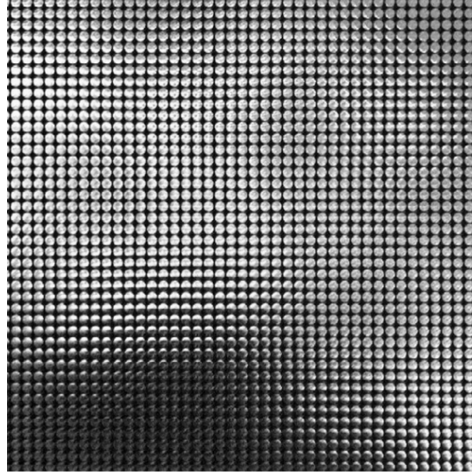
Cada una de estas imágenes con punto de vista ligeramente distinto se han generado con una apertura que es solo una fracción de la de la lente principal: esta disminución de la apertura afecta directamente al recorrido de enfoque que se puede obtener procesando la imagen plenóptica. El cómputo del *focal stack* brindará un conjunto de imágenes que aisladamente presentarán una profundidad de enfoque propia de la apertura de la lente principal, pero que en su conjunto barren la profundidad de campo equivalente a las subaperturas originadas por la inclusión de las microlentes.

Resumiendo, independientemente del orden que sigan, un par de dimensiones dan cuenta de resolución espacial mientras que las otras informan sobre la distribución angular de la función plenóptica. Para generar el *focal stack* se suman convenientemente muestras que, provenientes de detrás de una o varias microlentes, convergerían sobre un mismo punto dentro (y por lo tanto también fuera) del cuerpo de la cámara plenóptica.

4.7.2. Formación de imagen en $s = 0$

El caso más simple es conseguir la imagen que se formaría a la distancia de enfoque de una cámara convencional, esto es, rehacer el efecto de integración angular de la función plenóptica que propicia la lente principal pero

[5]Lo que vendría a ser, sobre el esquema *flatland*, representar la imagen siguiendo la ordenación de arriba a abajo de rayos en el mismo orden que llegan al sensor, a la izquierda. O de arriba a abajo, pero según el orden a la derecha, volviendo sobre el sensor, a la izquierda, y equivalentemente, sobre el conjugado de éste en espacio objeto, esto es, barriendo la escena desde distintos puntos de vista.



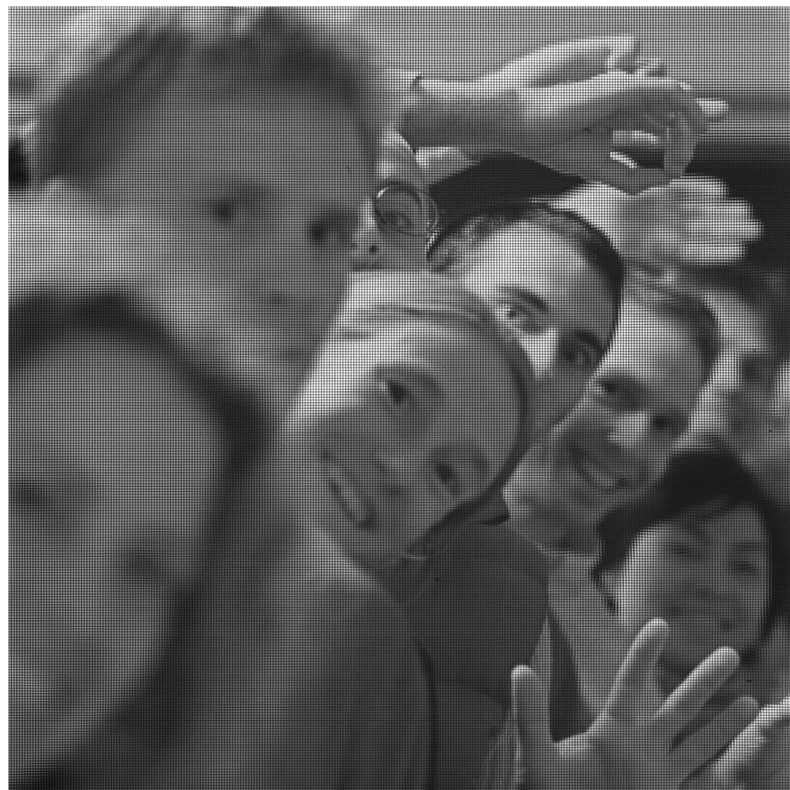


Figura 4.10: Detalle e imagen general captada con una plenóptica ordenada según el orden de píxeles sobre el sensor, lo que se puede interpretar como una macroimagen conteniendo las BRDFs del plano conjugado.

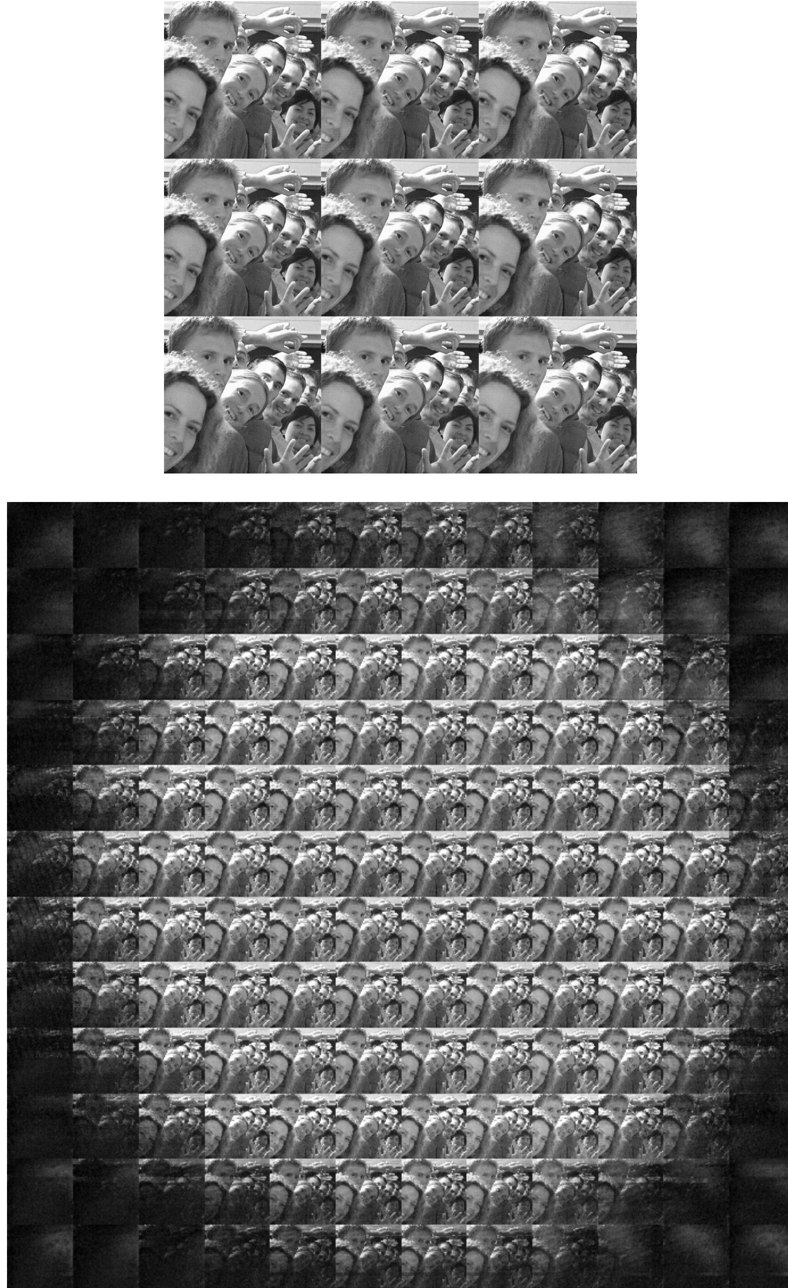


Figura 4.11: Detalle e imagen general captada con una plenóptica reordenada para seguir el orden de subaperturas sobre la apertura, dando lugar a un conjunto de imágenes con diferentes puntos de vista.

Figura 4.12: DOF comparados de una imagen de subapertura y de la imagen central del *focal stack*.

que habíamos logrado posponer con la inclusión de las microlentes. Para ello basta con sumar entre sí los píxeles tras cada microlente. El resultado es una imagen de resolución $N \times N$, cuyo *DOF* es el que cabría esperar en una cámara convencional con misma lente objetivo y sin microlentes. Es como si cada microlente se hubiera convertido en un macropíxel, con una SNR $M \times M$ veces mayor que el que había en cada píxel considerado aisladamente. Esto equivale al máximo sacrificio de resolución espacial, que decae consecuentemente $M \times M$ veces. El efecto, adelantado en los párrafos precedentes, se puede apreciar en la figura 4.12. A la izquierda se muestra una de las 12x12 imágenes de subapertura de la imagen plenóptica de la figura 4.11, y a su derecha el resultado de promediar todas las imágenes de subapertura. La profundidad de enfoque es máxima cuando la relación señal ruido es mínima, y viceversa. Para contrarrestar el efecto oscurecedor que supone en la imagen promediada la atenuación en las imágenes de subaperturas a los extremos, se muestran ambas imágenes con los histogramas ecualizados. Se observan patrones repetitivos, especialmente en horizontal, que son debidos al desalienamiento de los bordes de las microlentes y los píxeles tras éstas, que obliga a realizar cierta interpolación para obtener dimensiones enteras.

Este caso más simple, ¿cómo se contempla en el algoritmo de la *aFST*? Obsérvese la figura 4.8 en la página 120. A la izquierda se describe visualmente cómo son las líneas discretas en un problema 2D de tamaño 8x8. Así, para recorrer con pendiente $s = 0$ el plano, se suman los nodos con índices cubriendo por entero, de 0 a 7, la dimensión u, sin variar en ningún momento la otra dimensión. Esto se repetiría para cada posible desplazamiento d sobre esa otra dimensión. Pues bien, eso es lo que, trasladado al problema de reenfoque, hemos descrito anteriormente con las frases *«sumar entre sí todos los píxeles tras cada microlente»* y *«promediar todas las imágenes de*

subapertura»: para generar cada pixel, en la posición *(i, j)*, de la imagen de «pendiente 0» del *focal stack*, se recorren promediando entre sí las muestras del *lightfield* cuyas coordenadas posicionales están fijadas al valor de interés, *(i, j)*, mientras se consideran libres las de tipo angular. Esto sobre la figura 4.9 equivale a promediar entre sí los rayos que llegan a cada nodo a la izquierda, lo que representaba a una microlente. La virtud del algoritmo de la *aFST*, cabe recordar, es brindar conjuntamente con este caso más simple, la solución para todo el volumen de enfoque, esto es, para todas las combinaciones del parámetro s, que se identifica con la profundidad de enfoque y relaciona dimensiones posicionales con dimensiones angulares del *lightfield*, y hacerlo con el mínimo número de operaciones.

4.7.3. Formación de imagen combinando rayos de varias microlentes

Veamos un caso más interesante: cuando $s = 1$. La figura 4.8 nos indica que el comportamiento esperado para esa pendiente es el de recorrer hasta la mitad del dominio u sin ascender en la otra dimensión, y realizar el resto del recorrido un valor por encima del de partida. Esto interpretado a nivel de rayos, figura 4.13, implica aunar dos haces, que cruzan, la mitad, por una microlente y la otra mitad por la microlente vecina. Según sea vecina a uno u otro lado, estaremos en lo que en la sección 4.4, figura 4.6, denominamos diferentes «cuadrante de reenfoque» o, si se quiere, pendientes positivas y negativas.

Si para el caso $s = 0$ decíamos que el punto geométrico del cruce de rayos era exactamente sobre cada microlente (y su conjugado en espacio objeto), ahora vemos que, por la manera de actuar del método, ya no se garantiza el que los rayos coincidan en un único punto. Precisamente la *aFST* no insiste en forzar la coincidencia exacta a base de interpolar rayos, y aún con todo veremos que el *focal stack* que se obtenga como resultado será aceptable. Para apreciar la razón de que esto último se cumple, obsérvese que si consideramos el punto de cruce de los rayos más extremos, simbolizado en la figura con un rombo gris relleno, como el punto de reenfoque representante de este desplazamiento d, y procedemos idénticamente para los desplazamiento $d+1$ y $d-1$, obteniendo uniones de rayos que representamos por los rombos azul y verde, entonces los rayos aunados en los distintos desplazamientos cruzan por zonas excluyentes la línea perpendicular al eje óptico a la distancia en que se ubican los rombos, lo que se simboliza en la figura con los segmentos de línea rematados en flechas en torno a los rombos verde, gris y azul. Esto es, en lugar de forzar, por medio de interpolación, a que los rayos cruzen por el

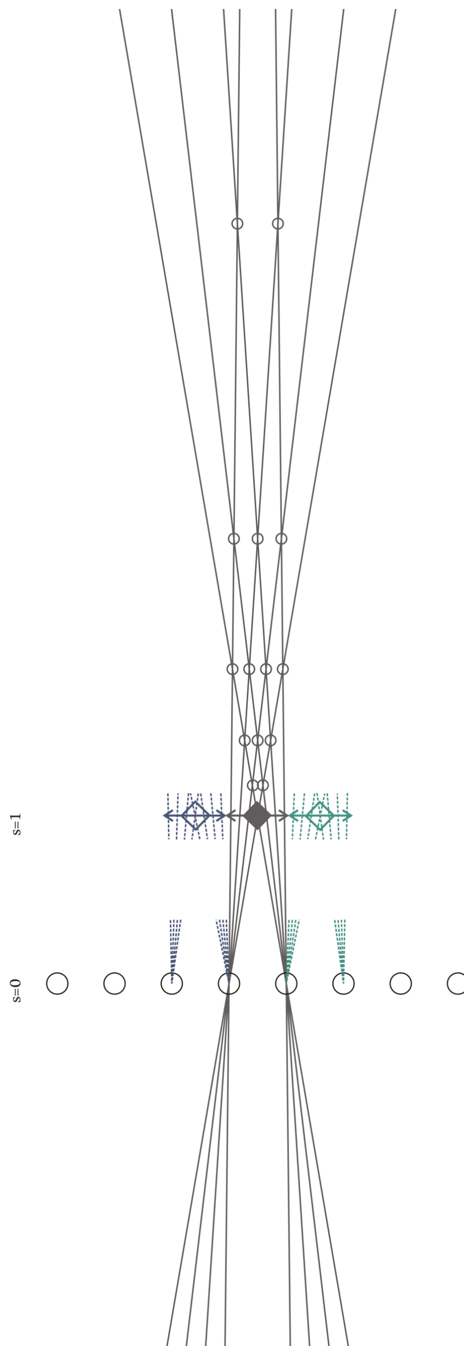


Figura 4.13: Posición geométrica de los rayos aunados por la *aFST* en el reenfoque con parámetro $s = 1$ sobre el problema de tamaño 8×8 de la figura 4.9. Se detalla en el texto.

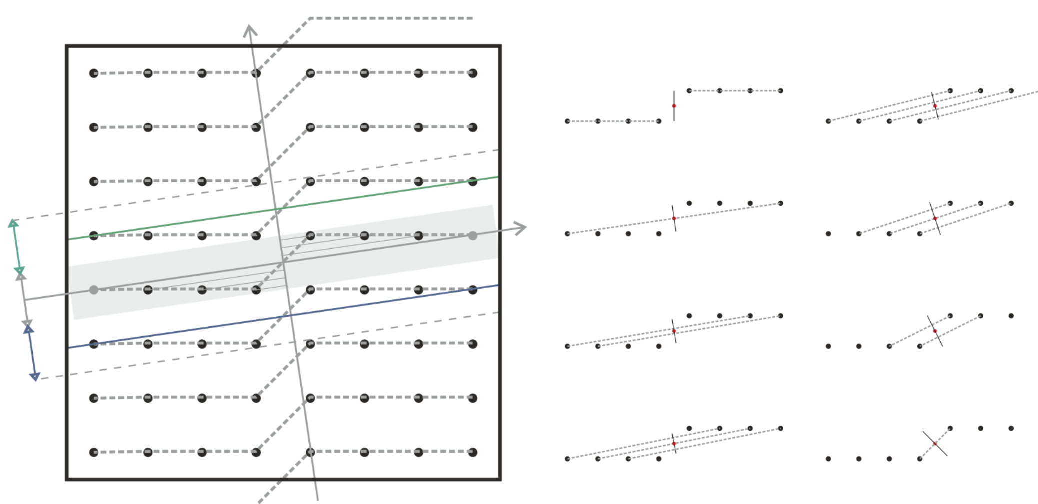


Figura 4.14: Zonas exclusivas en la 2D aDRT. Se detalla en el texto.

punto «representante», se reconoce que los rayos no cruzan exactamente por un punto, pero igualmente se les promedia, puesto que sí podemos asegurar que si a esa distancia hay objeto la selección de rayos del método está muestreando más o menos adecuadamente la zona en cuestión (y no introduce rayos indebidos). En conclusión, nos habremos ahorrado las ponderaciones en atención a cuánto se aproxima cada rayo en concreto al punto deseado.

Se puede apreciar esto mismo bajo otro punto de vista en la figura 4.14. Vemos a la izquierda el conjunto de muestras del *lightfield* de tamaño 8x8. Cada fila se corresponde con los rayos captados tras una misma microlente, y estos se representan dispuestos en horizontal dentro de cada fila atendiendo al ángulo de incidencia. Aunque no se ha representado se sobreentiende un eje horizontal de caracter angular y uno vertical de caracter posicional sobre el sensor. El operador de formación fotográfica equivale a una integral sobre las muestras angulares, y por lo tanto se puede entender, y es el caso $s = 0$, como la proyección de las muestras angulares sobre ese eje posicional, de tal manera que se obtienen 8 valores sobre el sensor, resultantes de promediar las 8 muestras en cada fila. El caso $s = 1$ es más interesante. En la zona sombreada en gris se ubican las 8 muestras que recorre la línea «discreta» que impone para esa pendiente y desplazamiento el método de transformada rápida multiescala, simbolizada con la línea gris de punteado grueso. La primera mitad de muestras angulares pertenecen a una microlente y la otra mitad a una microlente adyacente. Sí se representan un par de ejes ligeramente inclinados respecto al par original, pero que igualmente tiene por origen el centro de las 64 muestras. La inclinación de los ejes se corresponde con la inclinación de la línea que une las dos muestras a los extremos de la

línea discreta, círculos en gris, y que constituye un nuevo eje angular. Como hemos reiterado es la pendiente de dicho eje la que determina la distancia de enfoque en z. Perpendicular a éste está el eje posicional para la distancia fijada por la inclinación anterior. Se puede apreciar que solo las dos muestras a los extremos se proyectan sobre un mismo punto sobre este eje posicional, mientras que el resto de muestras se proyectan alrededor de dicho punto, pero no coincidentes. Ahora bien, lo que el método asegura es que si repitiéramos estas proyecciones para los conjuntos de muestras que se unen en el cómputo de los líneas discretas con desplazamientos inmediatamente anterior y posterior, representados con las líneas verde y azul paralelas al eje angular, cada conjunto de muestras se reparte una franja del eje posicional sobre la cual no se proyectan muestras de los conjuntos adyacentes. Los rayos que el método emplea para cubrir un desplazamiento (posición) y pendiente (distancia de enfoque) determinados son siempre los M rayos más adecuados.

Adicionalmente, nótese que además de las posiciones en que estos rayos atravesarán el plano de enfoque, se pueden también prever la distancia y posición exacta de los cruces de rayos en la figura 4.9, simbolizados con los círculos huecos más pequeños, atendiendo a la zona derecha de la figura 4.14[6]. Y es que estos cruces coinciden con las posibilidades de unir en línea recta dos o más muestras de las 8 que recorre cada línea discreta. En cada caso los cruces se producirán a la distancia que determine la pendiente de la(s) línea(s) de unión, habrá tantos cruces a esa distancia como líneas de unión paralelas se puedan dar y la posición en que se dan atiende al punto de proyección sobre el eje posicional de pendiente perpendicular a las líneas trazadas.

En la figura 4.15 se se ilustra el caso $s = 2$, en ella se representan los cruces de rayos por medio de círculos grises huecos. Se representa también, mediante los tres rombos verde, gris y azul, los puntos de cruce de los rayos más extremos de tres líneas discretas de desplazamientos consecutivos, y rodeando al rombo gris la zona exclusiva de paso sobre esa distancia de los rayos aunados para esa combinación, y que se muestran partiendo desde 3 microlentes a razón de 2, 4 y 2 rayos por microlente, tal como indica la figura 4.8 para $s = 2$.

Se representan como círculos grandes a las posiciones conjugadas de las microlentes y se etiquetan con la leyenda $\boldsymbol{s{=}0}$. Se representan, no ya los 3 de interés resaltados en la figura 4.13, sino todos los desplazamientos posibles a la distancia $\boldsymbol{s{=}1}$ y se puede apreciar que su número ha crecido hasta 9, desde los 8 posibles a la distancia $s = 0$. Esto se explica porque el método

[6]El último cruce previsto no aparece en la figura 4.9 porque se va fuera de la sección de interés representada.

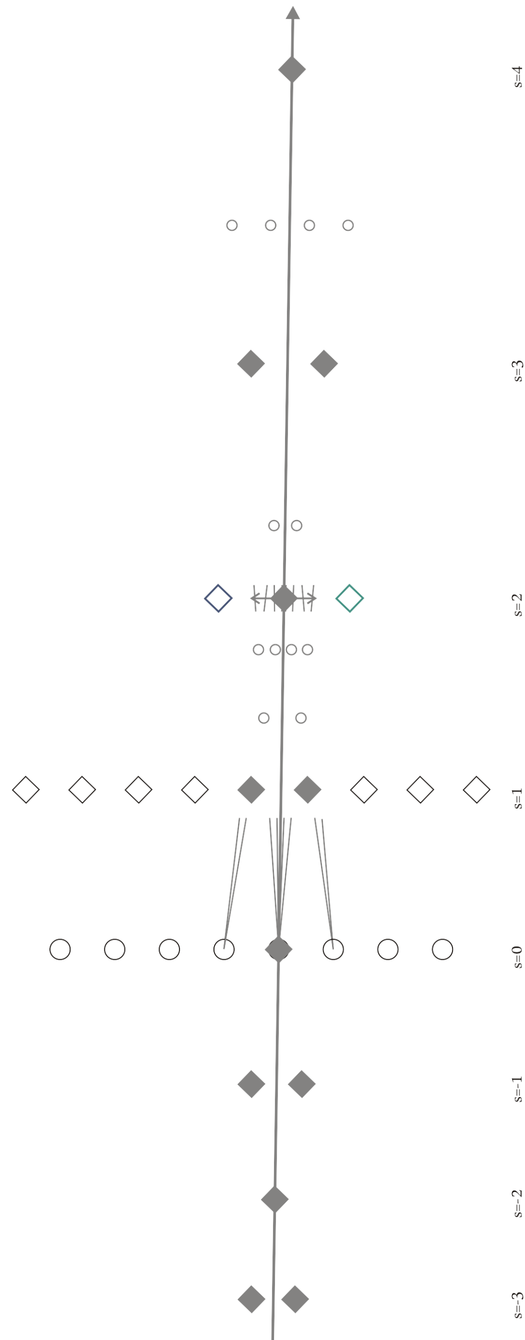


Figura 4.15: Posición geométrica de los rayos aunados por la *aFST* en el reenfoque con parámetro $s = 2$ sobre el problema de tamaño 8×8 de la figura 4.9. Se detalla en el texto.

incorpora rayos que provendrían de microlentes no contempladas físicamente en el sensor —pero que se quisieron simbolizar con las líneas discretas que deliberadamente exceden al recuadro del *lightfield* 8x8 de la figura 4.14—, simplemente contabilizando aportes de nula energía, que no otra cosa son los rellenos a cero que requiere el algoritmo. Esta misma razón explica la energía menor en los bordes de las imágenes del *focal stack*, a menos que se compense de antemano en razón del número de rayos efectivos que entran a computar en cada pixel del resultado. En esta compensación se pueden considerar otro efecto añadido: los aportes aminorados de los rayos en los bordes de las microlentes circulares. Todo esto se puede calibrar con un patrón de compensación que solo se precalcula una vez y no introduce complejidad computacional al algoritmo propuesto.

Si se contemplaran todas las posibilidades a la distancia ***s*=*2***, nos iríamos a 10 regiones sobre el sensor a esa distancia. Por el lado de pendientes negativas, la cuenta de regiones adicionales también se incrementa de esta manera: una más para ***s*=*-1***, dos para ***s*=*-2***, ...

Esta imagen también recoge que los píxeles del *focal stack* final están alineados para dos conjuntos disjuntos, formados por las imágenes a distancia par e impar. Así, un rayo a través del *focal stack*, simbolizado con línea gris gruesa, va cruzando sobre píxeles (rombos) en las distancias pares y entre píxeles en las impares. Por último nótese la disposición de los planos de reenfoque: progresivamente más distanciados según se incrementa la pendiente.

4.7.4. Cono discreto de desenfoque

Por último consideremos cómo se distribuye en el *focal stack* la luz recogida inicialmente tras una única microlente. Lo ilustramos con la figura 4.16, resultado de propagar la luz captada tras una microlente de una plenóptica con $16x16$ muestras, a razón de 1 unidad de luz por cada muestra. Esta figura se repetiría idénticamente en la otra mitad del *focal stack*. Se aprecia como el número de píxeles en el *focal stack* a los que alcanza la energía, inicialmente concentrada, aumenta paulatínamente, y cómo la energía (por razones obvias) no puede tomar valores «no enteros», por lo cual solo se cumple el ideal —que sería que la energía se dispersara de manera completamente homogénea en cada z— cuando el número de píxeles iluminados es potencia de 2. Esto es, en los planos que cumplen $1 + s = 2^k$, así cuando $s = 0$, $s + 1 = 2^0$ y tenemos un pixel de valor 256, cuando $s = 1$, 4 píxeles de valor 64, ... Lo obtenido, en este sentido, es lo mejor que se puede conseguir sin recurrir a interpolaciones. Además esta pirámide es invariante para una resolución angular dada.

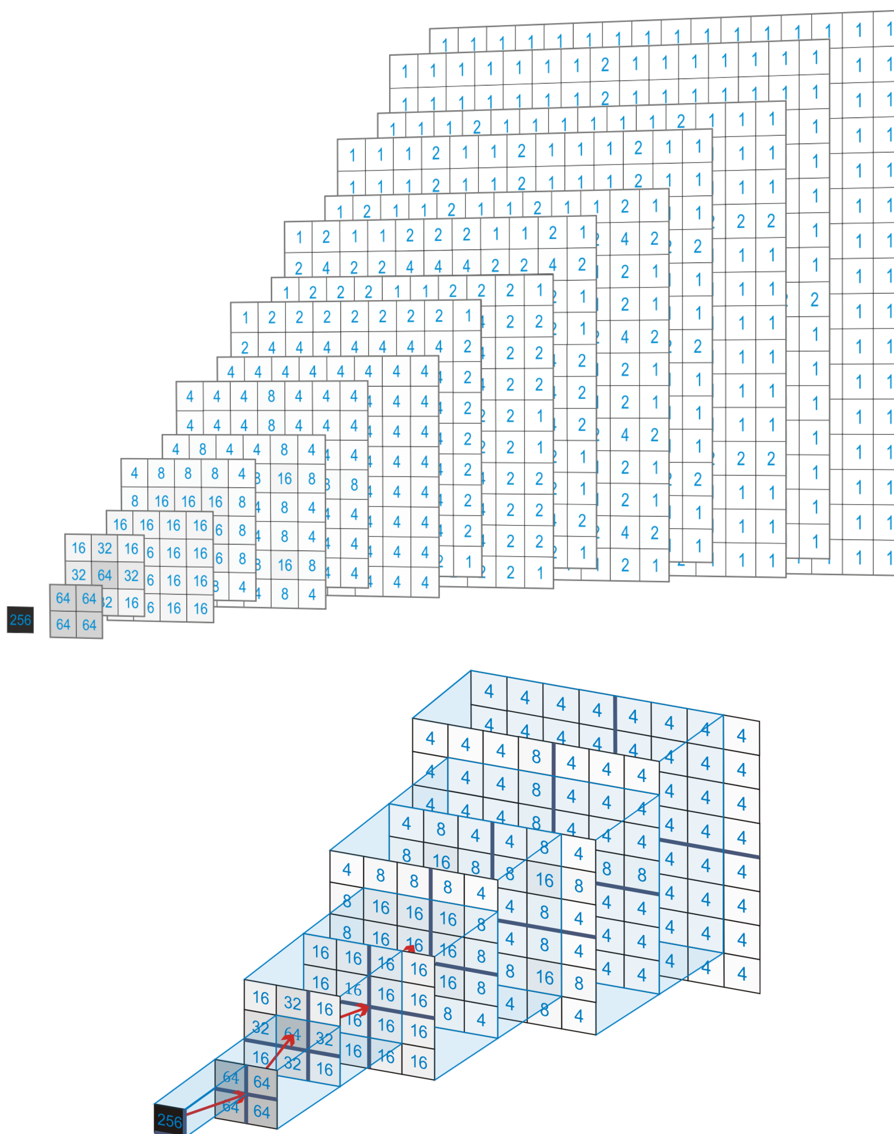


Figura 4.16: Pirámide de desenfoque discreto para una resolución angular de 16×16 rayos. Abajo, detalle de los primeros ocho planos. Nótese que un rayo que cruce el focal stack, según la interpretación geométrica que hemos visto, hay que buscarlo en posiciones alternantes en memoria según la z sea par o impar: así el rayo que atraviesa por 256, continúa en el segundo plano entremedias de los 4 valores 64, luego retoma el valor central 64, más allá entre 4 valores 16 por la cruz, etc.
Se obtiene una pirámide, en lugar de un cono, por usar microlentes rectangulares.

4.8. Implementación y rendimiento

Hemos implementado los algoritmos propuestos en *C++* y los hemos ejecutado en computadoras disponibles comercialmente (2.6 GHz, 1MB cache). Estas implementaciones no hacen uso de las extensiones de instrucciones multimedia, ni optimizaciones específicas de código, ni estrategias de aprovechamiento de cache. A pesar de esta falta de optimización hemos sido capaces de computar 31 planos de reenfoque a partir de un *lightfield* $256 \times 256 \times 16 \times 16$ en 2,09 segundos; y 23 planos a partir de un *lightfield* $292^2 \times 12^2$ en 2,33 segundos, haciendo uso del algoritmo para tamaños no potencia de dos (con la dimensión de lente descompuesta en la base $2 \times 3 \times 2$), o alternativamente, 2,68 segundos con el algoritmo potencia de dos y el *lightfield* extendido para ocupar $292^2 \times 16^2$.

Estos resultados contrastan favorablemente con los tiempos que Ren Ng da en [Ng 05b]: 47 segundos como tiempo de pre-proceso y adicionalmente entre 0,35 y 1,85 segundos por plano, dependiendo de la calidad del filtro de interpolación. Nuestro algoritmo se ejecuta unas diez veces más rápido.

Más aún, hemos implementado también nuestros algoritmos en CUDA [NVIDIA Corp. 07] y los hemos ejecutado sobre una tarjeta gráfica *NVIDIA GeForce 8800 GTX* (motor G80). El *lightfield* de tamaño $256^2 \times 16^2$ se procesó en 0,26 segundos, mientras que el de tamaño $292^2 \times 16^2$ llevó 0,36 segundos.

Se montaron dos tarjetas *NVIDIA 9800 GX2* sobre una única máquina de escritorio, cada una albergando dos motores GPU G92 y dos bloques independientes de 512MB de RAM. Se desarrolló una versión multi-GPU para esta plataforma, incrementando efectivamente el *throughput* a 4x el de una única G92, si bien el rendimiento de esta GPU individualmente, $0{,}38s$ para procesar el *lightfield* de $256^2 \times 16^2$, resultó inferior al logrado con la G80, lo que parece indicar que nuestra implementación en su forma actual está limitada por el rendimiento en los accesos a memoria, el cual es más bajo en el modelo *9800 GX2* que en la placa *8800 GTX*. En cualquier caso, estamos cerca del procesado en tiempo real para una vídeo cámara plenóptica a 2 mil líneas por frame y 24 frames por segundo con nuestra implementación multi-GPU.

Las imágenes en la figura 4.17 muestran la calidad del efecto de reenfoque que la transformada aproximada del *focal stack* puede lograr a partir de un *lightfield* capturado apropiadamente (la escena de ejemplo proviene de la cámara prototipada por Ren Ng, [Ng 05b], y se descargó de [Meng 07]). Las imágenes reenfocadas son difícilmente distinguibles, por inspección visual o numéricamente (menos de un 2.5 % de diferencia), de las obtenidas con los algoritmos en [Ng 05a] y [Nava 08], a pesar de la menor carga computacional de este método, lo cual hace especialmente apropiado a nuestro algoritmo

Figura 4.17: Fila intermedia: dos planos de enfoque a distinta distancia pertenecientes a un mismo *focal stack*, obtenido aplicando la *aFST* a un *lightfield* 4D. Fila superior: los mismos planos de enfoque obtenidos con la transformada basada en Fourier de la referencia [Nava 08]. Fila inferior: vista detallada de las imágenes de la fila intermedia, conteniendo regiones dentro y fuera de foco.

de cara a complementar a esos otros, en escenarios donde se desea una *preview* rápidamente y se permite el ligero sacrificio en calidad. O cuando el procesado offline no es una alternativa. También, la simplicidad del algoritmo propuesto es un punto a su favor a la hora de implementar una solución sobre un procesador embebido, μCPU, DSP o FPGA, al no requerir bloques de cómputo de FFT, y ni tan siquiera unidades de multiplicación.

Su mayor desventaja es la inferior calidad que se obtiene y la imposibilidad de reenfocar en más de $2N-1$ profundidades, las cuales además se disponen en posiciones fijas: que para un *lightfield* muestreado regularmente, se ubican muestreando el *focal stack* densamente comenzando en el plano conjugado a la pendiente más negativa, y luego progresivamente más espaciados a lo largo de z.

4.9. Consideraciones sobre la obtención de mapas de distancia

Con el fin de obtener una estimación de la distancia de la cámara a los objetos en la escena cabe la posibilidad de evaluar el volumen focal entendiendo dichos valores como la estimación del enfoque/desenfoque a lo largo de z para cada rayo (x, y).

En ese caso, para cada rayo, la mayor probabilidad de alcanzar una superficie se da a la distancia para la cual la medida del enfoque alcanza el valor máximo (o valor mínimo del desenfoque, dependiendo de cómo se plantée):

$$distancia(x, y) = \arg\max_z \Big(enfoque\big(volumenFocal(x, y, z) \big) \Big) \tag{4.24}$$

Ese sería un método puramente local, donde la estimación de la distancia para cada rayo del volumen focal no necesita el concurso de medidas en los rayos vecinos. Aunque siempre cabe la posibilidad de aplicar un suavizador de la solución local para atender a la restricción global de que las escenas típicamente no presentan cambios bruscos salvo en los bordes de los objetos.

Si aplicamos dicho método entendiendo el valor del volumen focal directamente como la medida del enfoque, es decir, si consideramos en la ec. (4.24) la función *enfoque* como innecesaria, *enfoque(x)* $= x$, los resultados obtenidos no pueden ser buenos, puesto que lo que se va a escoger como plano de máximo enfoque corresponderá en realidad al plano donde la intensidad del rayo fue máxima. Salvo que cambiemos la forma de trabajar del algoritmo para que, en lugar de promediar rayos, registre en cada caso la varianza de los rayos reunidos en los puntos evaluados, no tenemos un estimador válido

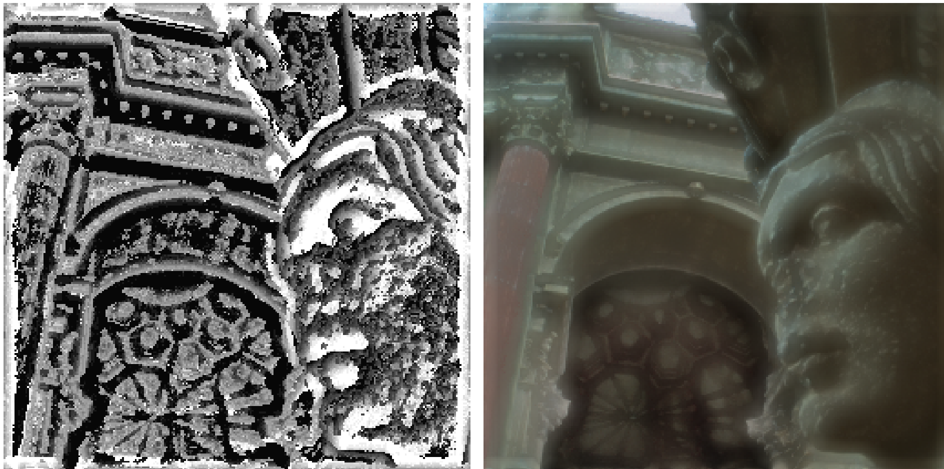


Figura 4.18: Mapa de distancias e imagen de máxima profundidad de campo usando como método de detección del enfoque el máximo valor del volumen focal.

del desenfoque. En cualquier caso seguiremos adelante con este método para ejemplificar un mal estimador de distancia.

Para verificar el método discutido y los que posteriormente propongamos procedemos a crear mapas de profundidad donde representaremos las distancias de los distintos rayos como una intensidad en escala de grises, asignando valores más claros a los objetos cercanos y más oscuros según nos alejemos. Haciendo uso de los valores de distancia se pueden crear imágenes donde se maximiza la profundidad de campo percibida, imágenes *all-in-focus*, por el método de fusionar todas las zonas detectadas como enfocadas en el volumen focal sobre una sola imagen. Cada zona de la escena solo se detecta como enfocada en un plano en concreto y son los píxeles de ese plano de reenfoque los que se eligen para la fusión.

Los resultados que se obtienen con esta manera de proceder, maximización del volumen focal, se recogen en la figura 4.18. Podemos observar que el mapa de distancias es claramente incorrecto y asigna a pixeles que pertenecen a la misma zona de la escena valores de distancia muy dispares. La imagen de profundidad de campo máximo presenta una especie de fulgor tamizado, precisamente por elegir los valores máximos del volumen focal.

Hemos de considerar más datos que los valores del volumen focal en el rayo de interés. En este caso a la hora de determinar la distancia más probable en el rayo (x, y) lo que pasamos a evaluar es un tubo de rayos en torno al de

interés:

$$distancia(x,y) = \arg\max_z \Bigg(enfoque\big(\underset{\substack{\chi\in[x-\frac{tam+1}{2},x+\frac{tam+1}{2}]\\ \psi\in[y-\frac{tam+1}{2},y+\frac{tam+1}{2}]}}{volumenFocal}(\chi,\psi,z)\big)\Bigg). \tag{4.25}$$

De esta manera decidir qué distancia asignar como más probable al rayo (x, y) implica evaluar ventanas de imagen en torno a dicho rayo en cada plano de reenfoque, quedándonos con el valor de z que maximize un cierto operador evaluado sobre las ventanas. Tal como se esquematiza en la figura 4.19.

4.9.1. Enfoque por contraste

Como primer método de evaluación del enfoque en ventanas en torno al rayo de interes proponemos considerar la máxima diferencia de color entre cualesquiera dos píxeles dentro de ventanas rectangulares de ancho $2{*}tam{+}1$. Esto es, en la ecuación (4.25), consideramos:

$$ventana_{x,y,tam}(z) = \underset{\substack{\chi\in[x-\frac{tam+1}{2},x+\frac{tam+1}{2}]\\ \psi\in[y-\frac{tam+1}{2},y+\frac{tam+1}{2}]}}{volumenFocal}(\chi,\psi,z), \tag{4.26a}$$

$$enfoque\Big(ventana_{x,y,tam}(z)\Big) = \max_{\substack{p\in ventana_{x,y,tam}(z),\\ q\in ventana_{x,y,tam}(z)-\{p\}}} |p-q|, \tag{4.26b}$$

$$distancia(x,y) = \arg\max_z \Bigg(enfoque\Big(ventana_{x,y,tam}(z)\Big)\Bigg) \tag{4.26c}$$

Probamos a variar el parámetro tam entre 1 y 10, y los resultados obtenidos se muestran en el cuadro 4.1. Se observa que el valor de $tam = 1$ ya conduce a una mejor solución que en el caso de estimador local de máximo del volumen focal, pero zonas que claramente deberían tomar el mismo valor de distancia aún difieren. Según se aumenta el valor de tam la solución es más uniforme, pero de aumentar en demasía, las zonas de transición entre planos de disparidad se desdibujan.

Otras zonas mal resueltas son aquellas que corresponden a zonas poco texturadas, donde por más que crece la ventana sigue sin concretarse el valor adecuado de distancia. En estas zonas, precisamente por tomar el color del rayo valores muy similares en todo el recorrido de z, la imagen de máxima profundidad de campo es correcta aún cuando el mapa denso de distancias no lo parezca.

Para el ejemplo mostrado el tamaño de ventana que mejor compromiso ofrece entre suavidad en los objetos y respeto de las zonas de borde, y mejor

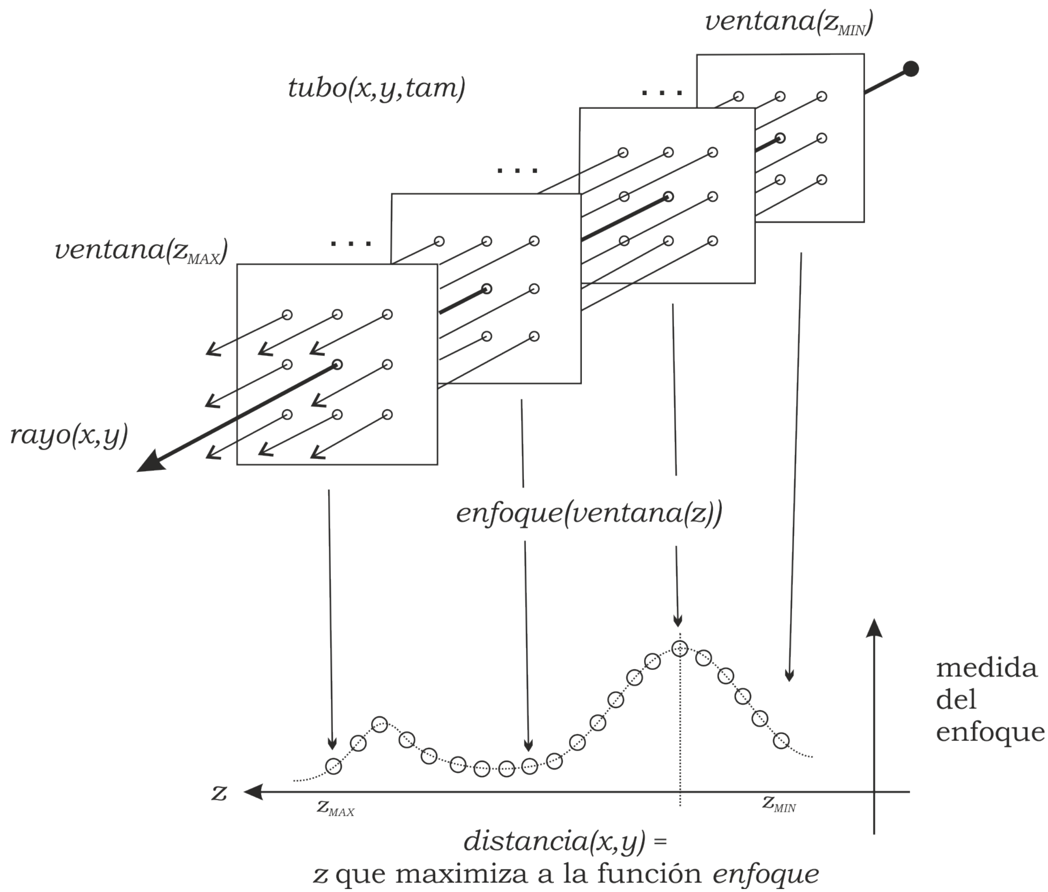
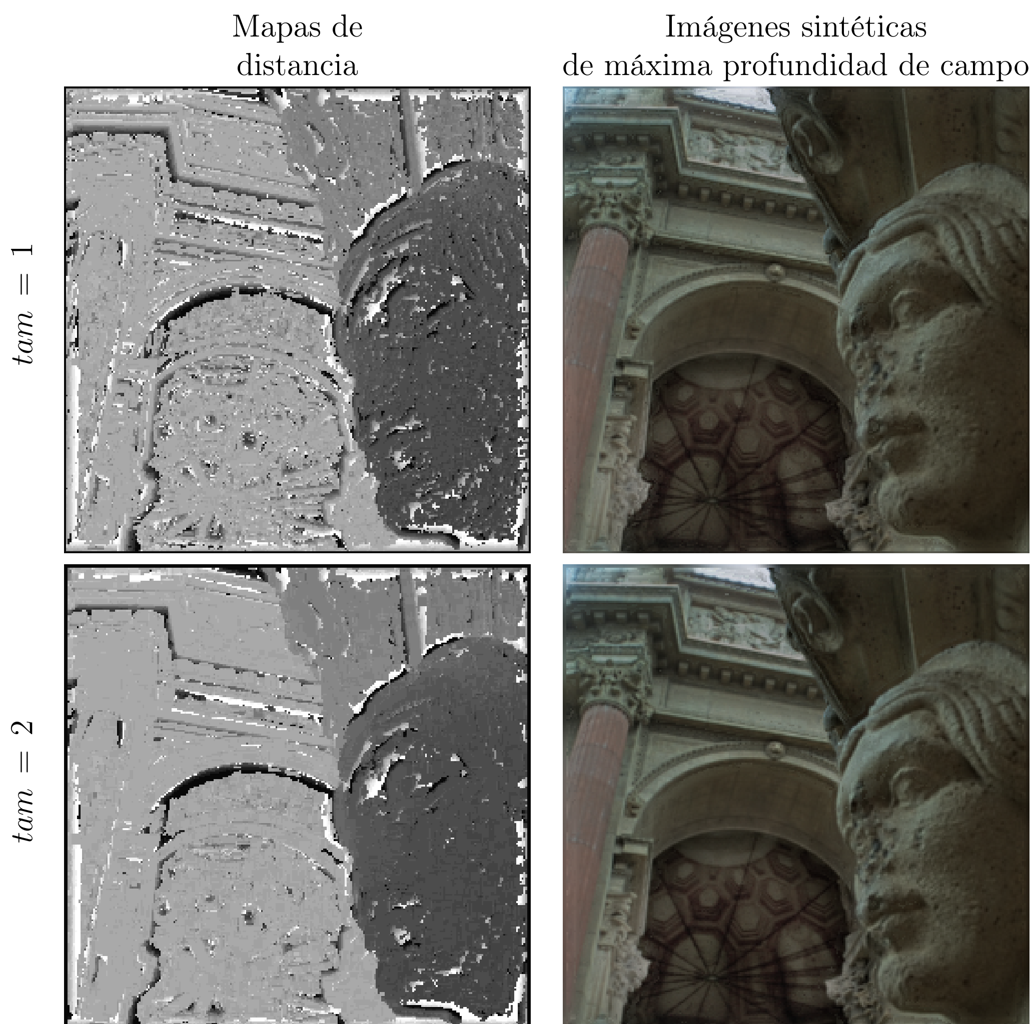
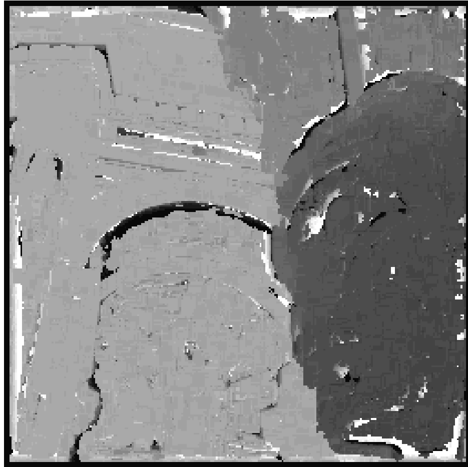
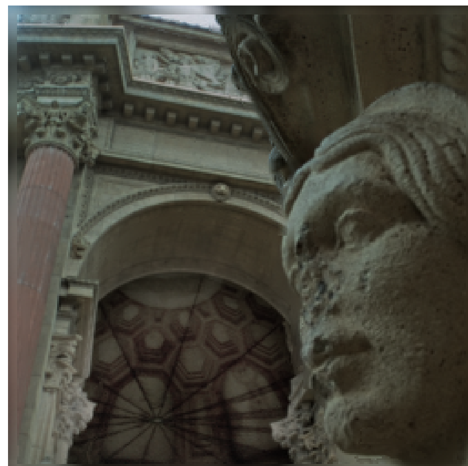


Figura 4.19: Un tubo de rayos en el entorno del rayo de interés (x, y). Evaluando un cierto operador sobre las ventanas de imagen —de ancho 3 en el ejemplo— por donde el tubo de rayos corta con cada plano del volumen focal, obtenemos una función del enfoque a lo largo de z para dicho tubo. La distancia desde la cámara a los objetos en la escena a través del rayo (x, y) es aquella que maximiza la medida de enfoque.

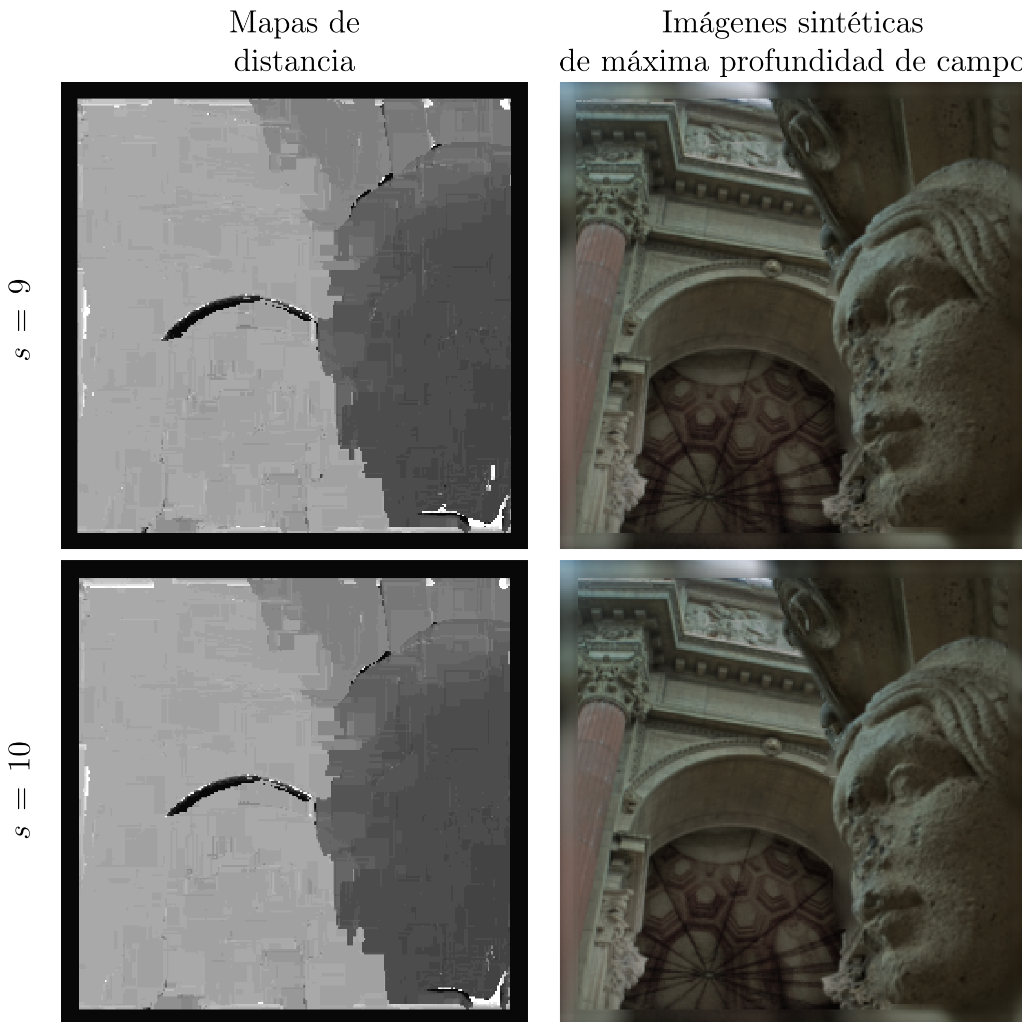
imagen sintética de máxima profundidad de campo genera, está entre los valores $tam = 3$ y $tam = 4$. Tamaños superiores a éstos presentan artefactos indeseados en los mapas de distancia.

Como lo que se evalúa es la distancia máxima entre dos píxeles de las ventanas —sin tener en cuenta al resto—, se puede dar el caso de que dos píxeles muy distanciados en color en un cierto plano sean los que primen en la elección de muchos rayos en la vecindad. Esta elección se extiende muchas veces hasta el límite impuesto por el tamaño de la ventana, con lo que en los mapas de distancia comienzan a aparecer zonas rectangulares claramente distinguibles alrededor de los píxeles de alto contraste. Para combatir este efecto indeseado proponemos un segundo método.



	Mapas de distancia	Imágenes sintéticas de máxima profundidad de campo
$tam = 3$		
$tam = 4$		
$tam = 5$		

	Mapas de distancia	Imágenes sintéticas de máxima profundidad de campo
$s = 6$		
$s = 7$		
$s = 8$	



Cuadro 4.1: Mapas de distancias e imágenes de máxima profundidad de campo usando como método de detección de enfoque el máximo contraste en ventanas de ancho $2 * tam + 1$ del volumen focal.

4.9.2. Enfoque evaluado mediante diferencias

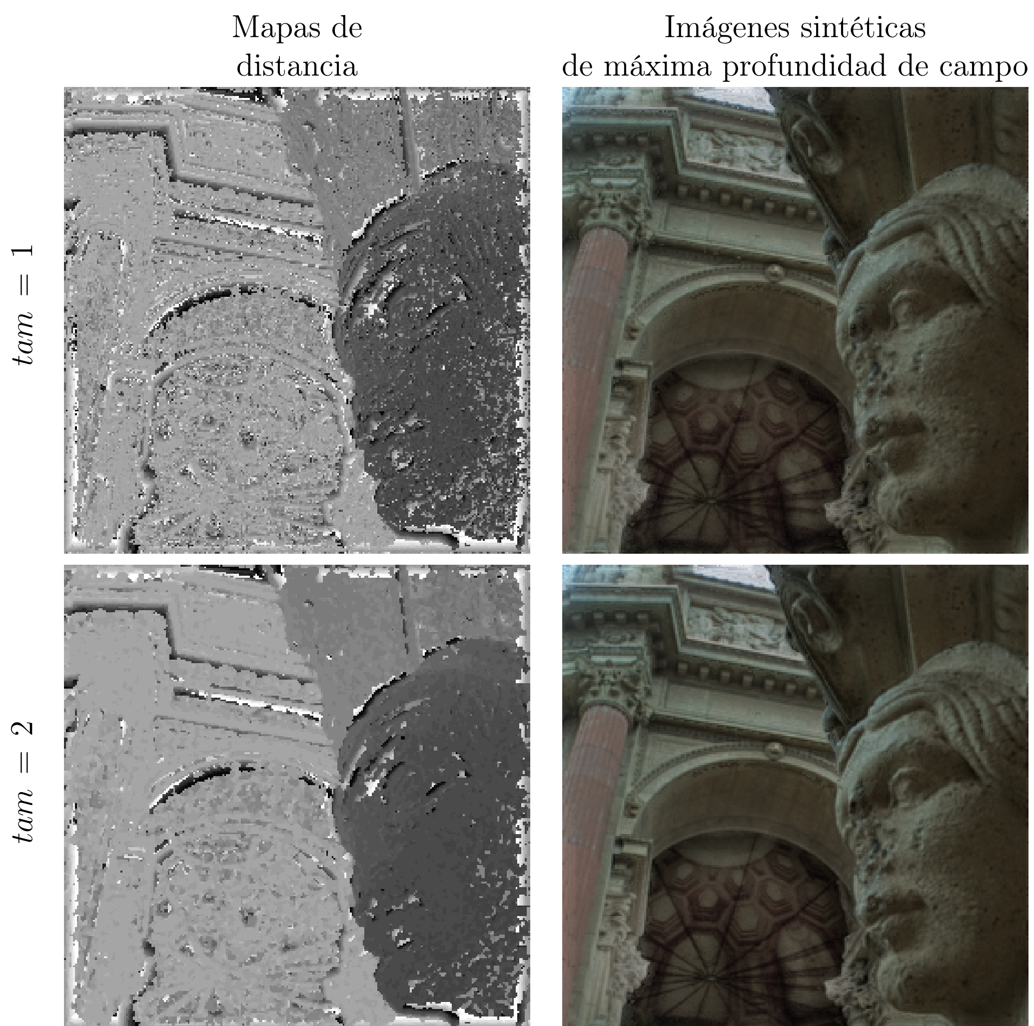
Consideremos qué sucede cuando los valores de todos los píxeles de la ventana influyen en el valor devuelto por la función enfoque. Para ello procedemos como en el método anterior, ecuaciones (4.26), pero modificamos la función enfoque, que queda:

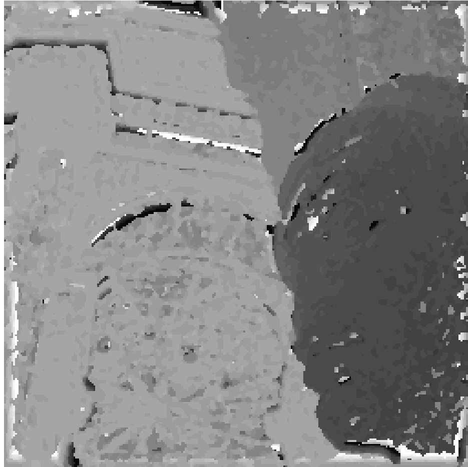
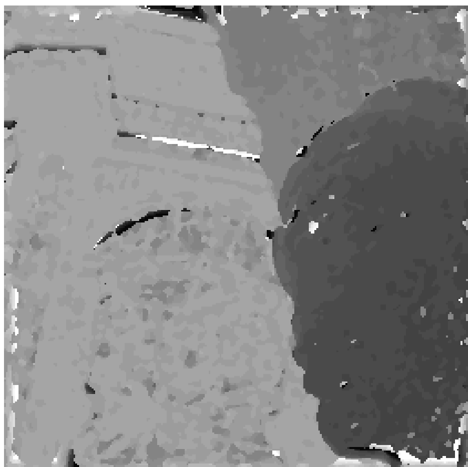
$$enfoque\Big(ventana_{x,y,tam}(z)\Big) =$$

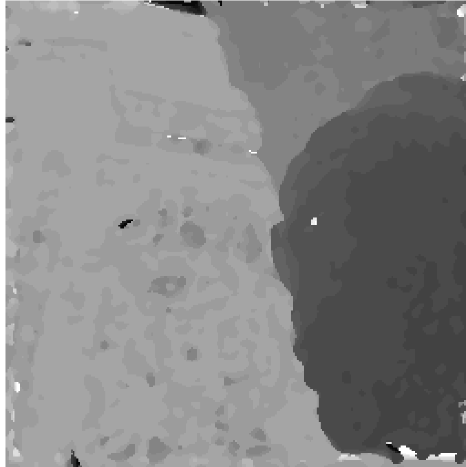
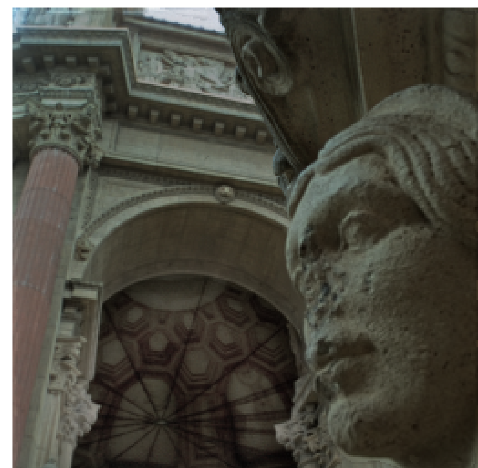
$$\sum\left(\left(\operatorname{diffx}\left(ventana_{x,y,tam}(z)\right)+\operatorname{diffy}\left(ventana_{x,y,tam}(z)\right)\right)\circledast G_{2tam+1}\right). \tag{4.27}$$

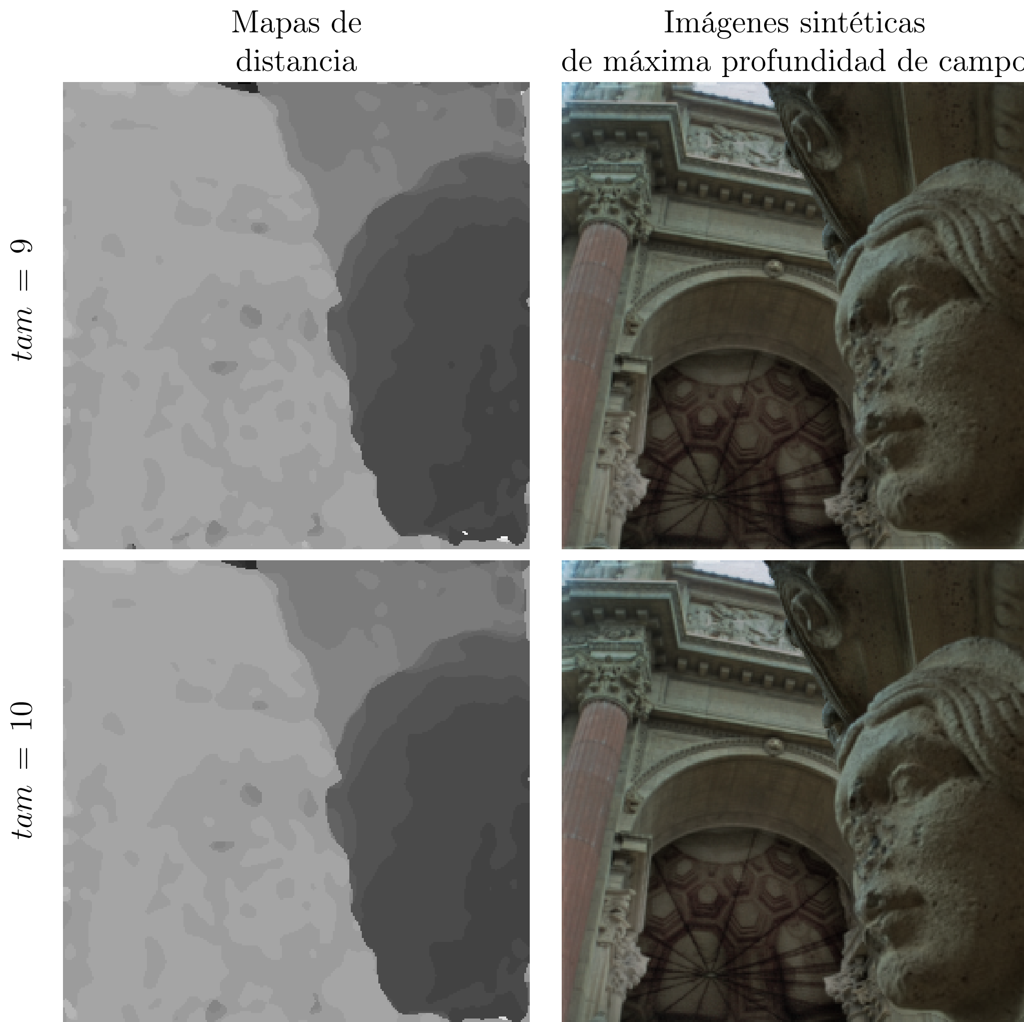
Donde los operadores diffx y diffy devuelven las diferencias absolutas de intensidad entre los píxeles de la ventana, bien sea en horizontal o en vertical, y llevamos a cabo una convolución con una ventana gaussiana bidimensional del mismo tamaño que los datos.

Los resultados de operar de esta manera son los mostrados en el cuadro 4.2. Ahora los resultados son más uniformes, y no hay traza de los artefactos rectangulares que estropeaban los mapas de distancias anteriormente. El tamaño más adecuado pasa a ser cuando $tam = 7$, pues es para este semiancho que todos los defectos han sido suavizados, mientras que los perfiles de los objetos se mantienen.



	Mapas de distancia	Imágenes sintéticas de máxima profundidad de campo
$tam = 3$		
$tam = 4$		
$tam = 5$		

	Mapas de distancia	Imágenes sintéticas de máxima profundidad de campo
$tam = 6$		
$tam = 7$		
$tam = 8$		



Cuadro 4.2: Mapas de distancias e imágenes de máxima profundidad de campo usando como método de detección de enfoque el valor acumulado de las diferencias de intensidad, en horizontal y vertical, entre píxeles de cada ventana, de ancho $2 * tam + 1$, del volumen focal.

En la figura 4.20 se muestran los mejores resultados para ambos métodos, que se dan para tamaños de ventana diferente. El segundo método propuesto resuelve la escena de ejemplo de manera satisfactoria salvo por los contornos de la barbilla de la estatua.

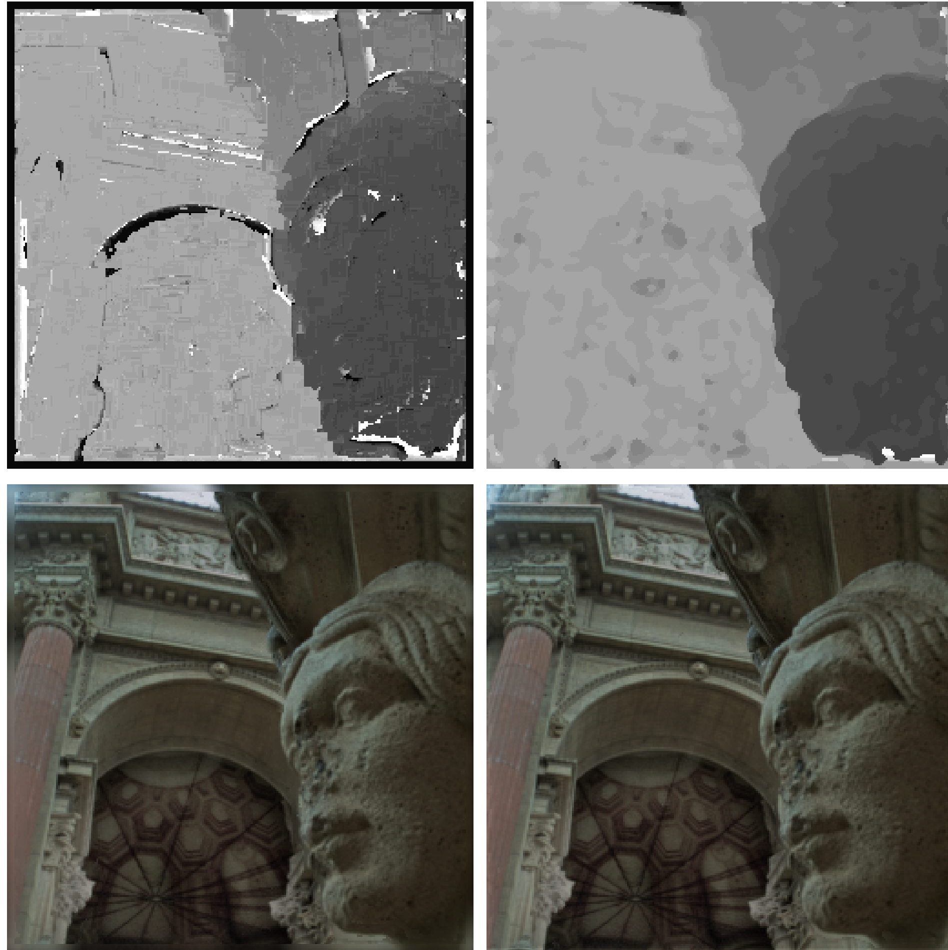


Figura 4.20: Mejores imágenes de máxima profundidad de campo para los métodos propuestos. A la izquierda método del contraste con $tam = 4$, a la derecha método de las diferencias con $tam = 7$.

Capítulo 5

Conclusiones

5.1. Sumario

- Se ha propuesto la implementación de diversos algoritmos de transformada rápida de Fourier adaptados al paradigma de *«stream processing»* sobre unidades gráficas de proceso: FFT sobre múltiples vectores 1D de menos de 4096 elementos, FFT sobre un vector 1D de más de 4096 elementos, FFT sobre un vector 2D de igual o diferente número de filas que de columnas.

  Con dichas implementaciones se logran reducir los tiempos de cómputo de las implementaciones más rápidas disponibles en ese momento: un 44 % de reducción de tiempo de cómputo respecto de la implementación en GPU de la librería BrookGPU, y 89 % de reducción de tiempo de cómputo respecto de la implementación FFTW corriendo sobre CPU, en ambos casos para vectores de números complejos de tamaño 1024×1024.

- Las implementaciones de la FFT sobre GPU se han aplicado al problema de la recuperación modal de la fase del frente de onda para sensores Shack-Hartmann, en una de las primeras aplicaciones efectivas de las GPUs en la literatura de óptica adaptativa.

- Se ha propuesto una modificación a los algoritmos de cómputo de la transformada discreta de Radon para posibilitar su extensión más allá de las dos dimensiones, y se han descrito los cambios necesarios para que trabajen sobre arrays de tamaños no potencia de 2.

- Se han derivado los algoritmos de transformada discreta de Radon que trabajan directamente sobre 3 y 4 dimensiones, sin recurrir a la propiedad de separabilidad.

- Se ha propuesto el algoritmo de la transformada aproximada de *«focal stack»* que, basándose en los algoritmos de la transformada discreta de Radon, computa directamente un volumen focal 3D a partir de una imagen plenóptica 4D.

  Los tiempos de cómputo con dicho algoritmo logran una reducción del tiempo de cómputo de más del 90 % respecto a los basados en la fotografía por *«slice de Fourier»*, los más rápidos hasta el momento, sin apenas merma en la calidad visual de los resultados obtenidos.

- Se ha implementado la transformada propuesta para que corra sobre múltiples GPUs, lo que permite el procesamiento en tiempo de adquisición de video.

- Se ha comprobado que el volumen focal permite extraer un mapa denso de distancia de la escena observada, evaluando cuál es el plano de mejor enfoque de todo el volumen para los distintos objetos.

5.2. Publicaciones

5.2.1. Contribuciones recogidas en esta Memoria

- J. G. Marichal Hernández, F. Rosa y J. M. Rodríguez Ramos. *Considerations on the FFT variants for an efficient stream implementation on GPU*. En 1st International Conference on Computer Vision Theory and Applications, Setúbal, Portugal, 2006.

- J. G. Marichal Hernández, F. Rosa y J. M. Rodríguez Ramos. *A multi output fast Fourier implementation on graphics hardware for optical simulation*. En International Mediterranean Modeling Multiconference, EMSS 2006, Barcelona, España, 2006.

- J. M. Rodríguez Ramos, J. G. Marichal Hernández y F. Rosa. *Modal Fourier wavefront reconstruction using graphics processing units*. En SPIE Astronomical Telescopes 06, Orlando, EEUU, 2006.

- J. G. Marichal-Hernández, F. Rosa F. Pérez Nava, Rene Restrepo y J. M. Rodríguez-Ramos. *An integrated system for Virtual Scene Rendering, Stereo Reconstruction and Accuracy Estimation*. En Geometric Modeling and Imaging 2006, Londres, Reino Unido, 2006.

- J. M. Rodríguez-Ramos, F. Rosa y J. G. Marichal-Hernández. *WAVEFRONT ABERRATION AND DISTANCE MEASUREMENT PHASE CAMERA*, 2007. Patente WO/2007/082975; PCT/ES2007/000046.

- J. G. Marichal-Hernández, J. M. Rodríguez-Ramos y F. Rosa. *Modal Fourier wavefront reconstruction using graphics processing units*. Journal of Electronic Imaging, vol. 16, no. 2, páginas 23005–9, 2007.

- F. Pérez Nava, J. P. Lüke, J. G. Marichal-Hernández, F. Rosa y J. M. Rodríguez-Ramos. *A simulator for the CAFADIS real time 3D TV camera*. En 3DTV-CON 08, Estambul, Turquía, 2008.

- J. M. Rodríguez-Ramos, J. G. Marichal-Hernández, F. Rosa y F. Pérez Nava. *METHOD AND CAMERA FOR THE REAL-TIME ACQUISITION OF VISUAL INFORMATION FROM THREE-DIMENSIONAL SCENES*, 2009. Patente WO/2009/090291; PCT/ES2009/000031.

- F. Rosa, J. G. Marichal-Hernández y J. M. Rodríguez-Ramos. *Transformada multiescala de Radon: más allá de las 2D y el particionado binario*. En URSI 09, Santander, España, 2009.

- J. G. Marichal-Hernández. *Desentrañando la FFT. Enseñanzas aplicables a la programación de GPUs.* En I Jornadas interdisciplinares de Computación y Comunicaciones, Universidad Politécnica de Valencia, España, 2009.

- J. G. Marichal-Hernández, F. Rosa J. P. Lüke, F. Pérez Nava y J. M. Rodríguez-Ramos. *Fast approximate focal stack transform.* En 3DTV-CON 09, Potsdam, Alemania, 2009.

- J. G. Marichal Hernández, F. Rosa J. P. Lüke y J. M. Rodríguez-Ramos. *Fast approximate 4D:3D discrete Radon transform, from light field to focal stack with O(N^4) sums.* En IS&T/SPIE Electronic Imaging 2011, San Francisco, 2011.

- J. G. Marichal Hernández, F. Rosa J. P. Lüke y J. M. Rodríguez-Ramos. *Fast approximate 4D:3D discrete Radon transform for light field refocusing.* Journal of Electronic Imaging, vol. 21, no. 2, *en imprenta*, 2012.

5.2.2. Otras contribuciones, en los mismos tópicos

- F. Rosa, J. G. Marichal-Hernández y J. M. Rodríguez-Ramos. *Wavefront phase recovery using Graphical Processing Units (GPU's).* En SPIE Europe Remote Sensing 2004, Maspalomas, España, 2004.

- J. G. Marichal-Hernández, L. F. Rodríguez-Ramos, F. Rosa y J. M. Rodríguez-Ramos. *Atmospheric wavefront phase recovery by use of specialized hardware: graphical processing units and field-programmable gate arrays.* Applied Optics, vol. 44, no. 35, páginas 7587–7594, 2005.

- J. G. Marichal Hernández, J. M. Rodríguez Ramos y F. Rosa. *Recuperación de fase de frente de onda atmosférico usando hardware gráfico.* En XI Congreso Nacional de Teledetección, Puerto de La Cruz, España, 2005.

- J. P. Lüke, F. Rosa, F. Pérez Nava, J. G. Marichal-Hernández y J. M. Rodríguez-Ramos. *Sistema de reconstrucción estéreo en tiempo real y su evaluación con ruido.* En URSI 07, Tenerife, España, 2007.

- J. M. Rodríguez-Ramos, E. Magdaleno, C. Domínguez Conde y J. G. Marichal-Hernández. *2D-FFT implementation on FPGA for wavefront phase recovery from the CAFADIS camera.* En SPIE Astronomical Imaging 2008, Marsella, Francia, 2008.

- I. Montilla, M. Reyes Garcia-Talavera, M. Le Louarn, J. G. Marichal-Hernández, J. M. Rodríguez-Ramos y L.F. Rodríguez-Ramos. *Performance of the Fourier transform reconstructor for the European Extremely Large Telescope.* En SPIE Astronomical Imaging 2008, Marsella, Francia, 2008.

- F. Pérez Nava, J. G. Marichal-Hernández y J. M. Rodríguez-Ramos. *The discrete Focal Stack transform.* En European Signal Processing Conference 2008, Lausana, Suiza, 2008.

- J. G. Marichal-Hernández, F. Rosa J. P. Lüke y J. M. Rodríguez-Ramos. *Reenfoque digital a resolución completa y creación de pares estéreo con la cámara CAFADIS.* En XXVI Simposio de la URSI (Unión Científica Internacional de Radio), Leganés, 2009.

- J. M. Rodríguez-Ramos, B. Femenía, I. Montilla, L. F. Rodríguez-Ramos, J. G. Marichal-Hernández, J. P. Lüke, J. J. Díaz R. López y Y. Martín. *The CAFADIS camera: a new tomographic wavefront sensor for Adaptive Optics.* En 1st Adaptive Optics for Extremely Large Telescopes conference, París, Francia, 2009.

- J. P. Lüke, F. Perez Nava, J. G. Marichal Hernández, J. M. Rodríguez Ramos y F. Rosa. *Near real time estimation of super-resolved depth and all-in-focus images from a plenoptic camera using graphic processing units (GPU).* International Journal of Digital Multimedia Broadcast, vol. 2010, páginas 942037–12, 2010.

- J. P. Lüke, J. G. Marichal-Hernández, F. Rosa y J. M. Rodríguez-Ramos. *A prototype of a real-time single lens 3D camera.* En International Conference on 3D Systems and Applications, Tokyo, Japón, 2010.

- J. P. Lüke, J. G. Marichal-Hernández F. Rosa y J. M. Rodríguez-Ramos. *Un prototipo para la reconstrucción 3D de la cámara CAFADIS.* En XXV Simposium Nacional de la Unión Científica Internacional de Radio (URSI 2010), Bilbao, 2010.

- J. M. Rodríguez-Ramos, J. P. Lüke, R. López, J. G. Marichal-Hernández, I. Montilla, J. Trujillo-Sevilla, B. Femenía, M. Puga, M. López, J. J. Fernández-Valdivia, F. Rosa, C. Dominguez-Conde, J. C. Sanluis y L. F. Rodríguez-Ramos. *3D imaging and wavefront sensing with a plenoptic objective.* En SPIE Defense, Security and Sensing, Orlando, EEUU, 2011.

- I. Montilla, J. P. Lüke, J. G. Marichal-Hernández, M. Puga y J. M. Rodríguez-Ramos. *Real-time measurement of the Na layer profile for tomographic reconstruction: experimental results and its application to the E-ELT case.* En Second Int. Conf. on Adaptive Optics for Extremely Large Telescopes, AO4ELTs 2, Victoria, Canada, 2011.

- M. López-Marrero, L. F. Rodríguez-Ramos, J. G. Marichal-Hernández y J. M. Rodríguez-Ramos. *Static telescope aberration measurement using lucky imaging techniques.* Experimental Astronomy, vol. 22, páginas 1–11, 2012.

5.2.3. Otras contribuciones, en procesado de señal

- J. P. Lüke, J. G. Marichal-Hernández, F. Rosa y J. Almunia. *Real time automatic detection of Orcinus orca vocalizations in a controlled environment.* Applied Acoustics, vol. 71, no. 8, páginas 771–776, 2010.

- J. P. Lüke, F. Rosa, J. G. Marichal-Hernández y J. Almunia. *Modelado paramétrico de señales bioacústicas como técnica de compresión y extracción de características.* En XXV Simposium Nacional de la Unión Científica Internacional de Radio (URSI 2010), Bilbao, 2010.

Apéndice A

Códigos de ejemplo

A.1. Código BrookGPU de la 2D FFT de tamaño $N \neq M$

```
// Codigos de los Kernel
kernel float2 multComplex(float2 A<>, float2 B<>) {
  float2 result;
  float2 negpos = {-1, 1};
  result = A.xx*B.xy+negpos*A.yy*B.yx;
  return result;
}

kernel float4 multComplex2(float4 A<>, float4 B<>) {
  float4 result;
  float4 negpos = {-1, 1, -1, 1};
  result = A.xxzz*B.xyzw+negpos*A.yyww*B.yxwz;
  return result;
}

// Mariposa en la dimension X
kernel void butterfly1D (float4 X<>, float2 W<>,
                         out float4 output<>) {
   float4 negpos4 = {1,1,-1,-1};
   float2 negpos2 = {-1,1};
   output = X.xyxy + negpos4*X.zwzw;
   output.zw = output.zz*W+output.ww*W.yx*negpos2;
}

// Mariposa en 2 dimensiones con multiples salidas
```

```
kernel void butterfly(float4 Xer<>, float4 Xor<>,
                      float4 WBC<>,
                      out float4 outXer<>,
                      out float4 outXor<>) {
   float4 negpos={1,1,-1,-1};
   float4 BBCC;
   float2 DD, WD;
   BBCC.xy=Xer.zw;
   BBCC.zw=Xor.xy;
   BBCC = multComplex2(BBCC, WBC);
   WD = multComplex(WBC.xy, WBC.zw);
   DD = multComplex(Xor.zw, WD);

   outXer = Xer.xyxy+BBCC.zwzw+(BBCC.xyxy+DD.xyxy)*negpos;
   outXor = Xer.xyxy-BBCC.zwzw+(BBCC.xyxy-DD.xyxy)*negpos;
}

// Kernel de recombinacion
kernel void combine (float4 X_er_ec<>, float4 X_er_oc<>,
                     float4 X_or_ec<>, float4 X_or_oc<>,
                     out float4 result1q<>,
                     out float4 result2q<>,
                     out float4 result3q<>,
                     out float4 result4q<>) {
  result1q.xy = X_er_ec.xy;
  result1q.zw = X_er_oc.xy;
  result2q.xy = X_er_ec.zw;
  result2q.zw = X_er_oc.zw;
  result3q.xy = X_or_ec.xy;
  result3q.zw = X_or_oc.xy;
  result4q.xy = X_or_ec.zw;
  result4q.zw = X_or_oc.zw;
}
  ...

// Declaracion de los streams
float4 X<N, M/2>;      // Datos
float4 outQ<N, M/2>;   // Resultados 1 dimensionales

float4 outXer<N/2, M/2>; // Res. intermedios, filas pares
float4 outXor<N/2, M/2>; // Res. intermedios, fil. impares

  ...
```

```
// Codigo de la transformada: llamadas a Kernels
precalculateTwiddle(); // Prepara WBQs[] y WBCs[]
streamRead(Xreverse, data_X); // Transfiere datos a GPU
bitReverse(Xreverse, br, X);  // INDEX BIT REVERSAL en GPU

// Q etapas de transformacion 1D en la dimension mayor
for (l = 1; l <= Q; l++) {
  butterfly1D(X, WBQ(l), outQ);
  combine1D(outQ, outQ.domain(int2(1, 0), int2(M/2+1, N)),
            X.domain(int2(0, 0), int2(M/4, N)),
            X.domain(int2(M/4, 0), int2(M/2, N)));
}
// etapas de transformacion 2D en el resto
for (l = Q+1; l <= R; l++) {
    butterfly(X, X.domain(int2(0,1), int2(M/2,N+1)),
              WBC(l),
              outXer, outXor);
    combine(outXer,
            outXer.domain(int2(1,0), int2(M/2+1, N/2)),
            outXor,
            outXor.domain(int2(1,0), int2(M/2+1, N/2)),
            X.domain(int2(0,     0), int2(M/4, N/2)),
            X.domain(int2(M/4,   0), int2(M/4, N/2))
            X.domain(int2(0,   N/2), int2(M/4, N))
            X.domain(int2(M/4, N/2), int2(M/2, N)));
}
// Devuelve los resultados a CPU
streamWrite(X, data_Y);
```

A.2. Código de la transformada aproximada del *focal stack*

```
/* ------------------------------------------------------------- */
tPixel *radonZpositive(tPixel *data, int N, tBase baseM) {

  int m;
  int j, k, s, t, rho;
  int dir_fmp1, dir_fm;
  tPixel *fm, *fmp1, *aux;
  int M, mbits;
  int tamPendiente, tamRecorrido;
  int td, sd, rd, msd, mtd, digitoEtapa;
  tPixel res;

  mbits = baseM.length;
  M = productorioBase(baseM);
  tamPendiente = 1;

  fm = (tPixel *)calloc(sizeof(tPixel), M*(N+2*M)*M*(N+2*M));
  memcpy(fm, data, sizeof(tPixel) * M*(N+2*M)*M*(N+2*M));

  fmp1 = (tPixel *)calloc(sizeof(tPixel),
                        M*(N+2*M)*M*(N+2*M)/baseM.d[0]);

  // for mbits etapas, variando entre 0 y mbits-1
  // cada una computa f^m+1 a partir de f^m
  for (m = 0; m < mbits; m++) {
    extraeMenosSignificativo(&digitoEtapa, &baseM);
    tamRecorrido = productorioBase(baseM);
    for (k = 0; k < N+M; k++) {
       for (t = 0; t < tamRecorrido; t++) {
        for (j = 0; j < N+M; j++) {
          for (s = 0; s < tamRecorrido; s++) {
            for (rho = 0; rho < tamPendiente; rho++) {
              for (rd = 0; rd < digitoEtapa; rd ++) {
                dir_fmp1 = (((((((((k
                                  * (tamRecorrido)) + t)
                                  * (N+2*M)) + j)
                                  * (tamRecorrido)) + s)
                                  * (tamPendiente)) + rho)
                                  * digitoEtapa) + rd);
```

```
                res = 0;
                for (td = 0; td < digitoEtapa; td ++) {
                  mtd = min(td, rd);
                  for (sd = 0;
                       ((k + td*rho + mtd) < (N+M))
                         &&
                       (sd < digitoEtapa);
                       sd ++) {
                    msd = min(sd, rd);
                    if ((j + sd*rho + msd) < (N+M)) {
                      dir_fm =
                            ((((((((((((k + td*rho + mtd)
                                     * (tamRecorrido)) + t)
                                     * digitoEtapa) + td)
                                     * (N+2*M)) + (j+sd*rho+msd))
                                     * (tamRecorrido)) + s)
                                     * digitoEtapa) + sd)
                                     * (tamPendiente)) + rho);
                      // lectura y suma de datos
                      res += fm[dir_fm];
                    }
                  } // for sd
                }  // for td
                //  escritura del resultado
                fmp1[dir_fmp1] = res;
              }  // for rd
            }  // for rho
          }  // for j
        }  // for s
      }  // for t
    }  // for k
    tamPendiente *= digitoEtapa;
    // intercambio de buffer
    aux = fm;
    fm = fmp1;
    fmp1 = aux;
  }  // for m, etapas

  free(fmp1);
  return fm;
}  // radonZpositive
```

```
/* ------------------------------------------------------------ */
tPixel *radonZnegative(tPixel *data, int N, tBase baseM) {

  int m;
  int j, k, s, t, rho;
  int dir_fmp1, dir_fm;
  tPixel *fm, *fmp1, *aux;
  int M, mbits;
  int tamPendiente, tamRecorrido;
  int td, sd, rd, msd, mtd, digitoEtapa;
  tPixel res;

  mbits = baseM.length;
  M = productorioBase(baseM);
  tamPendiente = 1;

  fm = (tPixel *)calloc(sizeof(tPixel), M*(N+2*M)*M*(N+2*M));
  memcpy(fm, data, sizeof(tPixel) * M*(N+2*M)*M*(N+2*M));

  fmp1 = (tPixel *)calloc(sizeof(tPixel),
                         M*(N+2*M)*M*(N+2*M)/baseM.d[0]);

  // for mbits etapas, variando entre 0 y mbits-1
  // cada una computa f^m+1 a partir de f^m
  for (m = 0; m < mbits; m++) {
    extraeMenosSignificativo(&digitoEtapa, &baseM);
    tamRecorrido = productorioBase(baseM);
    for (k = M; k < N+2*M; k++) {
       for (t = 0; t < tamRecorrido; t++) {
        for (j = M; j < N+2*M; j++) {
          for (s = 0; s < tamRecorrido; s++) {
            for (rho = 0; rho < tamPendiente; rho++) {
              for (rd = 0; rd < digitoEtapa; rd ++) {
                dir_fmp1 = (((((((((k
                                 * (tamRecorrido)) + t)
                                 * (N+2*M)) + j)
                                 * (tamRecorrido)) + s)
                                 * (tamPendiente)) + rho)
                                 * digitoEtapa) + rd);
                res = 0;
                for (td = 0; td < digitoEtapa; td ++) {
                  mtd = min(td, rd);
                  for (sd = 0;
```

```
                       ((k - td*rho - mtd) >= M)
                         &&
                        (sd < digitoEtapa);
                       sd ++) {
                    msd = min(sd, rd);
                    if ((j - sd*rho - msd) >= M) {
                      dir_fm =
                           ((((((((((((k - td*rho - mtd)
                                     * (tamRecorrido)) + t)
                                     * digitoEtapa) + td)
                                     * (N+2*M)) + (j-sd*rho-msd))
                                     * (tamRecorrido)) + s)
                                     * digitoEtapa) + sd)
                                     * (tamPendiente)) + rho);
                      // lectura y suma de datos
                      res += fm[dir_fm];
                    }
                  } // for sd
                }  // for td
                //  escritura del resultado
                fmp1[dir_fmp1] = res;
              }  // for rd
            }  // for rho
          }  // for j
        }  // for s
      }  // for t
    }  // for k
    tamPendiente *= digitoEtapa;
    // intercambio de buffer
    aux = fm;
    fm = fmp1;
    fmp1 = aux;
  }  // for m, etapas
  free(fmp1);
  return fm;
}  // radonZpositive




/* -------------------------------------------------------------- */
// El siguiente codigo obvia los detalles de inicializacion,
// recogida de argumentos, lectura y salvado
```

```
int main(int argc, char **argv) {
  /* Los tamaños del lightfield son NxNxMxM
     M se descompone en la base baseM */

  initial_size = sizeof(tPixel)*M*(N+2*M)*M*(N+2*M);
  final_size = sizeof(tPixel)*N*N*(2*M-1);

/* Lectura a memoria, se han extendido y rellenada con
   las dimensiones originalmente de tamanho N */
  struct data3 {unsigned short int r, g, b; };
  data3 *buf = (data3 *)malloc(sizeof(data3)*N*M);
  for (k = 0; k < N*M; k++) {
    fread(buf, sizeof(data3), N*M, fileInput);
    for (j = 0; j < N*M; j++) {
      data[((N+2*M)*M-M*M-1-k)*(N+2*M)*M + M*M + j].r =
                                          (float)buf[j].r;
      data[((N+2*M)*M-M*M-1-k)*(N+2*M)*M + M*M + j].g =
                                          (float)buf[j].g;
      data[((N+2*M)*M-M*M-1-k)*(N+2*M)*M + M*M + j].b =
                                          (float)buf[j].b;
    }
  }

  dataZp=radonZpositive(data, N, baseM);
  dataZn=radonZnegative(data, N, baseM);

}
```

Apéndice B

Derivación del mapeo entre etapas consecutivas de la *aFST*

La ecuación (4.20), de transformación parcial hasta la etapa m, expresada para la etapa $m+1$ se convierte en

$$\tilde{f}^{\,m+1}(\overbrace{\boldsymbol{s}\,|\,\mathbf{v_1}\,|\,\mathbf{v_2}}^{2n-m-1\ bits}\,|\,d_1\,|\,d_2) =$$

$$\tilde{f}^{\,m+1}(\overbrace{s_{n-m-1},\ \overbrace{s_{n-m},\ldots,s_{n-1}}^{\boldsymbol{\sigma}:\ m\ bits}}^{\boldsymbol{s}:\ m+1\ bits}\,|$$

$$\underbrace{v_{1_{m+1}},\ldots,v_{1_{n-1}}}_{\mathbf{v_1}:\ n-m-1\ bits}\,|\,\underbrace{v_{2_{m+1}},\ldots,v_{2_{n-1}}}_{\mathbf{v_2}:\ n-m-1\ bits}\,|\,d_1|\,d_2) =$$

$$\sum_{\mathbf{u_1}\in\mathbb{Z}_2^{m+1}}\ \sum_{\mathbf{u_2}\in\mathbb{Z}_2^{m+1}} f(\lambda(\underbrace{\mathbf{u_1},\ \mathbf{v_1}}_{n})|\ \lambda(\underbrace{\mathbf{u_2},\ \mathbf{v_2}}_{n})|$$

$$l^{m+1}_{\lambda(\boldsymbol{s})}(\mathbf{u_1}) + d_1|\ l^{m+1}_{\lambda(\boldsymbol{s})}(\mathbf{u_2}) + d_2) \quad \text{(B.1)}$$

En los índices $\mathbf{u_i}$ en ambos sumatorios distingamos la parte Más Significativa, u_{i_m}, del resto, $\boldsymbol{\mu_i}$.

$$\tilde{f}^{\,m+1}(s_{n-m-1},\ \boldsymbol{\sigma}\,|\,\mathbf{v_1}\,|\,\mathbf{v_2}\,|\,d_1\,|\,d_2) =$$

$$\sum_{\boldsymbol{\mu_1}\in\mathbb{Z}_2^m}\sum_{u_{1_m}\in\mathbb{Z}_2^1}\sum_{\boldsymbol{\mu_2}\in\mathbb{Z}_2^m}\sum_{u_{2_m}\in\mathbb{Z}_2^1} f(\lambda(\underbrace{\boldsymbol{\mu_1},u_{1_m},\mathbf{v_1}}_{n})|\ \lambda(\underbrace{\boldsymbol{\mu_2},u_{2_m},\mathbf{v_2}}_{n})|$$

$$l^{m+1}_{\lambda(s_{n-m-1},\boldsymbol{\sigma})}(\boldsymbol{\mu_1},u_{1_m}) + d_1|\ l^{m+1}_{\lambda(s_{n-m-1},\boldsymbol{\sigma})}(\boldsymbol{\mu_2},u_{2_m}) + d_2) \quad \text{(B.2)}$$

De acuerdo a nuestra definición de *líneas digitales*, ec. (4.6),

$$l^{m+1}_{\lambda(s_{ls},\boldsymbol{\sigma})}(\boldsymbol{\mu},u_{MS}) = l^{m}_{\lfloor\lambda(s_{ls},\boldsymbol{\sigma})/2\rfloor}(\boldsymbol{\mu}) + u_{MS}\left\lfloor\frac{\lambda(s_{ls},\boldsymbol{\sigma})+1}{2}\right\rfloor,$$

aplicando que $\left\lfloor \frac{\lambda(s_{ls},\boldsymbol{\sigma})}{2} \right\rfloor = \lambda(\boldsymbol{\sigma})$ y que $\left\lfloor \frac{\lambda(s_{ls},\boldsymbol{\sigma})+1}{2} \right\rfloor = s_{ls} + \lambda(\boldsymbol{\sigma})$, obtenemos:

$$l^{m+1}_{\lambda(s_{ls},\boldsymbol{\sigma})}(\boldsymbol{\mu}, u_{MS}) = l^{m}_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu}) + u_{MS}\Big(s_{ls} + \lambda(\boldsymbol{\sigma})\Big)$$

que es la fórmula que relaciona la definición de líneas digitales para dos tamaños de dominio consecutivos, 2^m y 2^{m+1}, poniendo de relieve la influencia de los índices más (MS) y menos (ls) significativos de la pendiente y la variable de integración. Podemos aplicar este resultado en (B.2), obteniendo

$$\begin{aligned}
&\tilde{f}^{m+1}(s_{n-m-1},\, \boldsymbol{\sigma} \,|\, \mathbf{v_1} \,|\, \mathbf{v_2} \,|\, d_1 \,|\, d_2) = \\
&\quad \sum_{\boldsymbol{\mu_1}\in\mathbb{Z}_2^m} \sum_{u_{1_m}\in\mathbb{Z}_2^1} \sum_{\boldsymbol{\mu_2}\in\mathbb{Z}_2^m} \sum_{u_{2_m}\in\mathbb{Z}_2^1} f(\lambda(\boldsymbol{\mu_1}, u_{1_m}, \mathbf{v_1})|\, \lambda(\boldsymbol{\mu_2}, u_{2_m}, \mathbf{v_2})| \\
&\qquad l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_1}) + u_{1_m}\big(s_{n-m-1} + \lambda(\boldsymbol{\sigma})\big) + d_1| \\
&\qquad\quad l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_2}) + u_{2_m}\big(s_{n-m-1} + \lambda(\boldsymbol{\sigma})\big) + d_2)
\end{aligned} \tag{B.3}$$

Si expandimos los sumatorios en u_{i_m}, obtenemos

$$\begin{aligned}
&\tilde{f}^{m+1}(s_{n-m-1},\, \boldsymbol{\sigma} \,|\, \mathbf{v_1} \,|\, \mathbf{v_2} \,|\, d_1 \,|\, d_2) = \\
&\quad \sum_{\boldsymbol{\mu_1}\in\mathbb{Z}_2^m} \sum_{\boldsymbol{\mu_2}\in\mathbb{Z}_2^m} f(\lambda(\boldsymbol{\mu_1}, 0, \mathbf{v_1})|\, \lambda(\boldsymbol{\mu_2}, 0, \mathbf{v_2})| \\
&\qquad l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_1}) + d_1|\, l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_2}) + d_2) + \\
&\quad \sum_{\boldsymbol{\mu_1}\in\mathbb{Z}_2^m} \sum_{\boldsymbol{\mu_2}\in\mathbb{Z}_2^m} f(\lambda(\boldsymbol{\mu_1}, 1, \mathbf{v_1})|\, \lambda(\boldsymbol{\mu_2}, 0, \mathbf{v_2})| \\
&\qquad l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_1}) + d_1 + s_{n-m-1} + \lambda(\boldsymbol{\sigma})|\, l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_2}) + d_2) + \\
&\quad \sum_{\boldsymbol{\mu_1}\in\mathbb{Z}_2^m} \sum_{\boldsymbol{\mu_2}\in\mathbb{Z}_2^m} f(\lambda(\boldsymbol{\mu_1}, 0, \mathbf{v_1})|\, \lambda(\boldsymbol{\mu_2}, 1, \mathbf{v_2})| \\
&\qquad l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_1}) + d_1|\, l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_2}) + d_2 + s_{n-m-1} + \lambda(\boldsymbol{\sigma})) + \\
&\quad \sum_{\boldsymbol{\mu_1}\in\mathbb{Z}_2^m} \sum_{\boldsymbol{\mu_2}\in\mathbb{Z}_2^m} f(\lambda(\boldsymbol{\mu_1}, 1, \mathbf{v_1})|\, \lambda(\boldsymbol{\mu_2}, 1, \mathbf{v_2})| \\
&\qquad l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_1}) + d_1 + s_{n-m-1} + \lambda(\boldsymbol{\sigma})|\, l^m_{\lambda(\boldsymbol{\sigma})}(\boldsymbol{\mu_2}) + d_2 + s_{n-m-1} + \lambda(\boldsymbol{\sigma}))
\end{aligned} \tag{B.4}$$

Cada sumando de la fórmula ya ha sido computado previamente, como valor de $\tilde{f}^{m}$: nótese que los sumatorios son sobre variables en $\mathbb{Z}_2^m$, y que las líneas son versiones de la función l^m. Por lo tanto, sustituyéndolos, llegamos

a la expresión deseada que mapea dos etapas parciales $\widetilde{f}^{\,m} \longrightarrow \widetilde{f}^{\,m+1}$:

$$\begin{aligned}\widetilde{f}^{\,m+1}(s_{n-m-1},\, \boldsymbol{\sigma} \,|\, \mathbf{v_1} \,|\, \mathbf{v_2} \,|\, d_1 \,|\, d_2) = \\ \widetilde{f}^{\,m}(\boldsymbol{\sigma}|\, 0, \mathbf{v_1}|\, 0, \mathbf{v_2}|\, d_1|\, d_2) + \\ \widetilde{f}^{\,m}(\boldsymbol{\sigma}|\, 1, \mathbf{v_1}|\, 0, \mathbf{v_2}|\, d_1 + s_{n-m-1} + \lambda(\boldsymbol{\sigma})|\, d_2) + \\ \widetilde{f}^{\,m}(\boldsymbol{\sigma}|\, 0, \mathbf{v_1}|\, 1, \mathbf{v_2}|\, d_1|\, d_2 + s_{n-m-1} + \lambda(\boldsymbol{\sigma})) + \\ \widetilde{f}^{\,m}(\boldsymbol{\sigma}|\, 1, \mathbf{v_1}|\, 1, \mathbf{v_2}|\, d_1 + s_{n-m-1} + \lambda(\boldsymbol{\sigma})|\, d_2 + s_{n-m-1} + \lambda(\boldsymbol{\sigma}))\end{aligned} \quad \text{(B.5)}$$

Bibliografía

[Adelson 91] E. H. Adelson y J. R. Bergen. *The Plenoptic Function and the Elements of Early Vision.* M. Landy and J. A. Movshon, (eds) *Computational Models of Visual Processing*, 1991.

[Adelson 92] Edward H. Adelson y John Y. A. Wang. *Single Lens Stereo with a Plenoptic Camera.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 14, no. 2, páginas 99–106, 1992.

[Akeley 01] Kurt Akeley y Pat Hanrahan. *CS448A: Real-Time Graphics Architectures*, 2001. http://www.graphics.stanford.edu/courses/cs448a-01-fall/.

[Akeley 07] Kurt Akeley y Pat Hanrahan. *CS448: Real-Time Graphics Architectures*, 2007. https://graphics.stanford.edu/wikis/cs448-07-spring/.

[AMD-ATI 07] AMD-ATI. *AMD Stream Computing: Software Stack*, Dec. 2007. http://ati.amd.com/technology/streamcomputing/.

[AMD-ATI 08] AMD-ATI. *FireStream 9250: Breaking the 1 teraflop barrier*, 2008. http://ati.amd.com/technology/streamcomputing/product_firestream_9250.html.

[ATI 03] ATI. *Hardware image processing using ARB_fragment_program*, 2003. http://www.ati.com/developer/sdk/RadeonSDK/Html/Samples/OpenGL/HW_Image_Processing.html.

[Averbuch 01] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli y J. Waldén. *Fast Slant Stack: A Notion of Radon Transform for Data in a Cartesian Grid which is Rapidly computable, Algebraically Exact, Geometrically Faithful and invertible.* Informe técnico, Standford University, 2001.

[Averbuch 03] A. Averbuch y Y. Shkolnisky. *3D Fourier based discrete Radon transform.* Applied and Computational Harmonic Analysis, vol. 15, páginas 33–69(37), July 2003.

[Averbuch 08a] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli y Y. Shkolnisky. *A Framework for Discrete Integral Transformations I—The Pseudopolar Fourier Transform.* SIAM Journal on Scientific Computing, vol. 30, no. 2, páginas 764–784, 2008.

[Averbuch 08b] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli, Y. Shkolnisky y I. Sedelnikov. *A Framework for Discrete Integral Transformations II—The 2D Discrete Radon Transform.* SIAM Journal on Scientific Computing, vol. 30, no. 2, páginas 785–803, 2008.

[Bailey 91] David H. Bailey y Paul N. Swarztrauber. *The Fractional Fourier Transform and Applications.* SIAM Review, vol. 33, no. 3, páginas 389–404, Sept 1991.

[Ballard 82] Dana H. Ballard y Cristopher M. Brown. Computer vision. Prentice Hall, Englewood Cliffs (New Jersey), 1982. Capítulo 4.

[Beylkin 87] G. Beylkin. *Discrete Radon transform.* IEEE Trans. Acoust., Speech, Signal Processing, vol. 35, no. 2, páginas 162–172, 1987.

[Bluestein 70] L. I. Bluestein. *A linear filtering approach to the computation of the discrete Fourier transform.* IEEE Trans. Audio Electroacoust., vol. AU-18, páginas 451–455, Apr. 1970.

[Bolles 87] R. C. Bolles y H. H. Baker. *Epipolar-Plane Image Analysis: A Technique for Analyzing Motion Sequences.* En M. A. Fischler y O. Firschein, editores, Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, páginas 26–36. Kaufmann, Los Altos, CA., 1987.

[Bracewell 56] R. N. Bracewell. *Strip integration in radio astronomy.* Aust. J. Phys., vol. 9, páginas 198–217, 1956.

[Brady 92] Martin L. Brady y Whanki Yong. *Fast Parallel Discrete Approximation Algorithms for the Radon Transform.* En SPAA, páginas 91–99, 1992.

[Brady 98] Martin L. Brady. *A Fast Discrete Approximation Algorithm for the Radon Transform.* SIAM J. Comput., vol. 27, no. 1, páginas 107–119, 1998.

[Brandt 99] A. Brandt y J. Dym. *Fast Calculation of Multiple Line Integrals.* SIAM J. Sci. Comput., vol. 20, no. 4, páginas 1417–1429, 1999.

[Buck 03a] I. Buck y P. Hanrahan. *Data parallel computation on graphics hardware.* Informe técnico, Stanford University, 2003. Graphics Hardware 2003 Panel, `http://hci.stanford.edu/cstr/reports/2003-03.pdf`.

[Buck 03b] Ian Buck. *Brook Spec v0.2.* Stanford University, October 2003. `http://merrimac.stanford.edu/brook/`.

[Buck 04] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston y Pat Hanrahan. *Brook for GPUs: stream computing on graphics hardware.* ACM Trans. Graph., vol. 23, no. 3, páginas 777–786, 2004. `http://graphics.stanford.edu/projects/brookgpu/`.

[Buck 05] Ian Buck. *Stream Computing on Graphics Hardware.* Tesis doctoral, Stanford University, January 2005. `http://graphics.stanford.edu/~ianbuck/thesis.pdf`.

[Buehler 01] Chris Buehler, Michael Bosse, Leonard Mcmillan, Steven Gortler y Michael Cohen. *Unstructured lumigraph rendering*. En SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, páginas 425–432, New York, NY, USA, 2001. ACM Press.

[Burger 04] D. Burger y J.R. Goodman. *Billion-transistor architectures: there and back again*. Computer, vol. 37, no. 3, páginas 22–28, Mar 2004.

[Can 04] Canon. *Canon EF Lens Work III*, 4 edition, Feb 2004. http://www.canon-europe.com/Support/Documents/digital_slr_educational_tools/en/ef_lens_work_iii_en.asp.

[Chai 00] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan y Heung-Yeung Shum. *Plenoptic sampling*. En SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, páginas 307–318, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Chan 00] Shing-Chow Chan y Heung-Yeung Shum. *A Spectral Analysis for Light Field Rendering*. En 7th IEEE Intn'l Conference on Image Processing (ICIP 2000), Vancouver, Canada, September 2000.

[Cooley 65] James W. Cooley y John W. Tukey. *An Algorithm for the Machine Calculation of Complex Fourier Series*. Mathematics of Computation, vol. 19, páginas 297–301, April 1965.

[Dally 03] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonte, J-H A., N. Jayasena, U. J. Kapasi, A. Das, J. Gummaraju y I. Buck. *Merrimac: Supercomputing with streams*. En SC'03, Phoenix, Arizona, November 2003. http://merrimac.stanford.edu/.

[Dipert 05] Brian Dipert. *Instigating a platform tug of war*. EDN, Oct. 2005. http://www.edn.com/article/CA6262535.html.

[Dongarra 02] J. Dongarra. *Basic Linear Algebra Subprograms Technical Forum Standard*. En International Journal of High Performance Applications and Supercomputing, volumen 16, 1, páginas 1–111, 2002.

[Fatahalian 04] K. Fatahalian, J. Sugerman y P. Hanrahan. *Understanding the efficiency of GPU algorithms for matrix-matrix multiplication*. En HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, páginas 133–137, New York, NY, USA, 2004. ACM Press.

[Fernando 03] Randima Fernando y Mark J. Kilgard. The Cg tutorial: The definitive guide to programmable real-time graphics. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Fife 06] K. Fife, A. El Gamal y H.-S. P. Wong. *A 3D Multi-Aperture Image Sensor Architecture*. En Proceedings of the IEEE Custom Integrated Circuits Conference, páginas 281–284, September 2006.

[Frigo 05] M. Frigo y S.G.Johnson. *The design and implementation of FFTW3*. En Proceedings of the IEEE, volumen 93, páginas 216– 231, 2005. `http://www.fftw.org`.

[Georgiev 06] Todor Georgiev, Ke Colin Zheng, Brian Curless, David Salesin, Shree Nayar y Chintan Intwala. *Spatio-Angular Resolution Tradeoff in Integral Photography*. En Proceedings of Eurographics Symposium on Rendering, 2006. `http://grail.cs.washington.edu/projects/lfcamera/`.

[Georgiev 07a] Todor Georgiev y Chintan Intwala. *Light Field Camera Design for Integral View Photography*. Informe técnico, Adobe Systems Incorporated, 2007. `http://www.tgeorgiev.net/IntegralView.pdf`.

[Georgiev 07b] Todor Georgiev, Chintan Intwala y Derin Babacan. *Light-field capture by multiplexing in the frequency domain*. Informe técnico, Adobe Systems

Incorporated, 2007. http://www.tgeorgiev.net/FrequencyMultiplexing.pdf.

[Georgiev 08] Todor Georgiev, Chintan Intwala, Derin Babacan y Andrew Lumsdaine. *A Unified Frequency Domain Analysis of Lightfield Cameras.* En Proceedings European Converence on Computer Vision, 2008. To appear.

[Ghuloum 07] Anwar Ghuloum, Eric Sprangle, Jesse Fang, Gansha Wu y Xin Zhou. *Ct: A Flexible Parallel Programming Model for Tera-scale Architectures*, October 2007. http://techresearch.intel.com/articles/Tera-Scale/1514.htm.

[Gortler 96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski y Michael F. Cohen. *The lumigraph.* En SIGGRAPH '96, páginas 43–54, New York, NY, USA, 1996. ACM Press.

[Götz 96] W.A. Götz y H.J. Druckmüller. *A fast digital Radon transform - an efficient means for evaluating the Hough transform.* Pattern Recognition, vol. 29, no. 4, páginas 711–718, April 1996.

[Govindaraju 06] Naga Govindaraju, Scott Larsen, Jim Gray y Dinesh Manocha. *Efficient memory model for scientific algorithms on graphics processors.* Informe técnico, University of North Carolina at Chapell-Hill, 2006. GPUFFTW: High Performance Power-of-Two FFT Library using Graphics Processors, http://www.cs.unc.edu/~geom/GPUFFTW/.

[GPGPU 02] GPGPU. *General Purpose Computation Using Graphics Hardware*, 2002. http://www.gpgpu.com/.

[Gray 03] Kris Gray. Microsoft directx 9 programmable graphics pipeline. Microsoft Press, 2003.

[Gu 97] Xianfeng Gu, Steven J. Gortler y Michael F. Cohen. *Polyhedral Geometry and the Two-Plane*

Parameterization. En Julie Dorsey y Phillipp Slusallek, editores, Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering), páginas 1–12, New York, NY, 1997. Springer Wien.

[Gurrea 01] Emilio Camahort Gurrea. *4D Light-Field Modeling and Rendering.* Tesis doctoral, The University of Texas at Austin, May 2001.

[Harris 03] Mark J. Harris. *Real-Time Cloud Simulation and Rendering.* Tesis doctoral, University of North Carolina, 2003. `http://www.markmark.net/dissertation/`.

[Hartenstein 03] Reiner W. Hartenstein. *Data-Stream-Based Computing: Models and Architectural Resources.* En International Conference on Microelectronics, Devices and Materials (MIDEM 2003), Ptuj, Slovenia, 2003.

[Hartley 04] R. I. Hartley y A. Zisserman. Multiple view geometry in computer vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Hartmann 00] J. Hartmann. *Bemerkungen über den Bau und die Justirung von Spektrographen.* Z. Instrumentenkd, vol. 20, no. 47, 1900.

[Hetch 01] Eugene Hetch. Optics. Addison Wesley, 4 edition, August 2001.

[Hillis 86] W. Daniel Hillis y Jr. Guy L. Steele. *Data parallel algorithms.* Commun. ACM, vol. 29, no. 12, páginas 1170–1183, 1986.

[Horn 05] Daniel Horn. *libGPUFFT*, 2005. `http://sourceforge.net/projects/gpufft/`.

[Hu 05] Zhong Hu y Honghui Wan. *A novel generic fast Fourier transform pruning technique and complexity analysis.* Signal Processing, IEEE Transactions

on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on], vol. 53, no. 1, páginas 274–282, Jan. 2005.

[Hudgin 77] R.H. Hudgin. *Wave-Front Reconstruction for Compensated Imaging.* J. Opt. Soc. Am. A, vol. 67, páginas 375–378, 1977.

[Isaksen 00] Aaron Isaksen. Dynamically reparameterized light fields. Tesis de Master, Massachusetts Institute of Technology, Nov. 2000.

[ITRS 03] ITRS. *International Technology Roadmap for Semiconductors*, 2003. `http://www.itrs.net/Links/2003ITRS/Home2003.htm`.

[Ives 03] F. Ives. Parallax stereogram and process of making same. Patent US 725567, April 1903. `http://www.google.com/patents?id=ouBYAAAAEBAJ`.

[Jansen 04] T. Jansen, B. von Rymon-Lipinski, N. Hanssen y E. Keeve. *Fourier Volume Rendering on the GPU Using a Split-Stream-FFT.* En Proceedings of the Vision, Modeling and Visualization Workshop VMV'04, páginas 395–403. IOS Press BV, Nov 16-18 2004.

[Kaplan 08] Ken Kaplan. *Show Me The New Reality.* Intel Developer's Forum, Shangai, `http://blogs.intel.com/technology/2008/04/video_show_me_the_new_reality.php`, April 2008.

[Khailany 01] B. Khailany, W.J. Dally, U.J. Kapasi, P. Mattson, J. Namkoong, J.D. Owens, B. Towles, A. Chang y S. Rixner. *Imagine: media processing with streams.* Micro, IEEE, vol. 21, no. 2, páginas 35–46, Mar/Apr 2001.

[Kilgard 04] Mark J. Kilgard, editor. NVIDIA OpenGL extension specifications. NVIDIA Corporation, 2004. `http://developer.nvidia.com/object/nvidia_opengl_specs.html`.

[Knudsen 93] K.S. Knudsen y L.T. Bruton. *Recursive pruning of the 2D DFT with 3D signal processing applications.* Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on], vol. 41, no. 3, páginas 1340–1356, Mar 1993.

[Kolmogorov 05] Vladimir Kolmogorov y Martin Wainwright. *On the optimality of tree-reweighted max-product message passing.* En 21st Conference on Uncertainty in Artificial Intelligence (UAI), 2005.

[Krüger 03] Jens Krüger y Rüdiger Westermann. *Linear algebra operators for GPU implementation of numerical algorithms.* ACM Trans. Graph., vol. 22, no. 3, páginas 908–916, 2003.

[Levoy 96] Marc Levoy y Pat Hanrahan. *Light field rendering.* En SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, páginas 31–42, New York, NY, USA, 1996. ACM Press.

[Lin 00] Zhouchen Lin y Heung-Yeung Shum. *On the number of samples needed in light field rendering with constant-depth assumption.* Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on, vol. 1, páginas 588–595 vol.1, June 2000.

[Lindholm 01] Erik Lindholm, Mark J. Kilgard y Henry Moreton. *A user-programmable vertex engine.* En SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, páginas 149–158, New York, NY, USA, 2001. ACM Press.

[Lindholm 08] Erik Lindholm, John Nickolls, Stuart Oberman y John Montrym. *NVIDIA Tesla: A Unified Graphics and Computing Architecture.* IEEE Micro, vol. 28, no. 2, páginas 39–55, 2008.

[Lippmann 08] Gabriel Lippmann. *Epreuves reversible donnant la sensation du relief.* J. Phys., no. 7, páginas 821–825, 1908. Traducido al inglés en http://graphics.csail.mit.edu/~fredo/tmp/Lippmann.pdf.

[London 04] Barbara London, John Upton, Jim Stone, Ken Kobré y Betsy Brill. Photography. Prentice Hall, 8 edition, April 2004.

[Louis 83] A.K. Louis y F. Natterer. *Mathematical problems of computerized tomography.* Proceedings of the IEEE, vol. 71, no. 3, páginas 379–389, March 1983.

[Luebke 08] David Luebke y Steven Parker. *Interactive Ray Tracing With CUDA*. Siggraph Exhibitor Tech Sessions, August 2008. NVIDIA Corporation.

[Lumsdaine 08] Andrew Lumsdaine y Todor Georgiev. *Full Resolution Lightfield Rendering.* Informe técnico, Adobe Systems Incorporated, Jan. 2008. http://www.tgeorgiev.net/FullResolution.pdf.

[Macedonia 03] M. Macedonia. *The GPU enters computing's mainstream.* Computer, vol. 36, no. 10, páginas 106–108, Oct. 2003.

[Marichal-Hernández 05] J. G. Marichal-Hernández, L. F. Rodríguez-Ramos, F. Rosa y J. M. Rodríguez-Ramos. *Atmospheric wavefront phase recovery by use of specialized hardware: graphical processing units and field-programmable gate arrays.* Applied Optics, vol. 44, páginas 7587–7594, 2005.

[Markel 71] J. Markel. *FFT pruning.* Audio and Electroacoustics, IEEE Transactions on, vol. 19, no. 4, páginas 305–311, Dec 1971.

[McCool 04] Michael McCool, Stefanus Du Toit, Tiberiu Popa, Bryan Chan y Kevin Moule. *Shader algebra.* En SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, páginas 787–795, New York, NY, USA, 2004. ACM. http://libsh.org/.

[Meng 07] J. Meng. *GPU implementation of Light-Field Photography*, 2007. http://www.cs.virginia.edu/~jm6dg/.

[Microsoft 98] Microsoft. *DirectX*, 1998. http://www.microsoft.com/directx.

[Mitchell 03] Jason L. Mitchell, Marwan Y. Ansari y Evan Hart. Shaderx 2 - shader tips and tricks with DirectX 9, capítulo Advanced Image Processing with DirectX 9 Pixel Shaders. Wordware, 2003. Editado por Engel, Wolfgang.

[Monteyne 08] M. Monteyne. *RapidMind Multi-Core Development Platform.* Informe técnico, RapidMind Inc., Feb. 2008. http://www.rapidmind.net/.

[Montilla 08] I. Montilla, M. Reyes, M. Le Louarn, J. G. Marichal-Hernández, J. M. Rodríguez-Ramos y L. F. Rodríguez-Ramos. *Performance of the Fourier Transform Reconstructor for the European Extremely Large Telescope.* En SPIE Remote Sensing, volumen 7015, página 70152Y. SPIE, 2008.

[Moreland 03] Kenneth Moreland y Edward Angel. *The FFT on a GPU.* En Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, páginas 112–119. Eurographics Association, 2003.

[Nava 08] Fernando Pérez Nava, J. G. Marichal-Hernández y J. M. Rodríguez-Ramos. *The discrete Focal Stack transform.* En Proceedings of Eusipco'08, August 25–29 2008.

[Ng 05a] Ren Ng. *Fourier slice photography.* ACM Trans. Graph., vol. 24, no. 3, páginas 735–744, 2005.

[Ng 05b] Ren Ng, Mark Levoy, Mathieu Brédif, Gene Duval, Mark Horowitz y Pat Hanrahan. *Light Field Photography with a Hand-Held Plenoptic Camera.* Computer Science Tech Report CSTR 2005-02, Stanford University, April 2005.

[Ng 06] Ren Ng. *Digital Light Field Photography.* Tesis doctoral, Stanford University, 2006 2006.

[Nussbaumer 82] Henri J. Nussbaumer. Fast fourier transform and convolution algorithms. Springer-Verlag, second edition, 1982.

[NVIDIA Corp. 07] NVIDIA Corp. *CUDA Technology*, 2007. http://www.nvidia.com/CUDA.

[Olano 98] Marc Olano y Anselmo Lastra. *A shading language on graphics hardware: the pixelflow shading system.* En SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, páginas 159–168, New York, NY, USA, 1998. ACM Press.

[OpenGL-Apple 08] OpenGL-Apple. *OpenCL: Open Computing Language*, 2008. http://en.wikipedia.org/wiki/OpenCL.

[OpenGL ARB 01] OpenGL ARB. *Texture combiners: ARB_texture_env_combine*, 2001. http://oss.sgi.com/projects/ogl-sample/registry/ARB/texture_env_combine.txt.

[OpenGL ARB 02] OpenGL ARB. *Vertex programs: ARB_vertex_program*, 2002. http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_program.txt.

[OpenGL ARB 03] OpenGL ARB. *Extensiones GLSL: ARB_shader_objects, ARB_vertex_shader y ARB_fragment_shader*, 2003. http://oss.sgi.com/projects/ogl-sample/registry/ARB/shader_objects.txt,vertex_shader.txt,fragment_shader.txt.

[OpenGL ARB 04] OpenGL ARB. *Multiple Render Targets: ARB_draw_buffers*, 2004. http://oss.sgi.com/projects/ogl-sample/registry/ARB/draw_buffers.txt.

[Owens 05a] John D. Owens. GPU gems 2 - programming techniques for high-performance graphics and general-purpose computation, capítulo Streaming architectures and technology trends, páginas 457–470. Addison Wesley, 2005. Editado por Pharr, Matt.

[Owens 05b] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn y Timothy J. Purcell. *A Survey of General-Purpose Computation on Graphics Hardware*. En Eurographics 2005, State of the Art Reports, páginas 21–51, Agosto 2005.

[Owens 05c] John D. Owens, Shubhabrata Sengupta y Daniel Horn. *Assessment of Graphic Processing Units (GPUs) for Department of Defense (DoD) Digital Signal Processing (DSP) Applications*. Informe técnico ECE-CE-2005-3, IDAV, University of California–Davis, 2005. `http://www.idav.ucdavis.edu/publications/print_pub?pub_id=866`.

[Patt 97] Y.N. Patt, S.J. Patel, M. Evers, D.H. Friendly y J. Stark. *One billion transistors, one uniprocessor, one chip*. Computer, vol. 30, no. 9, páginas 51–57, Sep 1997.

[Pease 68] Marshall C. Pease. *An Adaptation of the Fast Fourier Transform for Parallel Processing*. J. ACM, vol. 15, no. 2, páginas 252–264, 1968.

[Poyneer 02] Lisa A. Poyneer, Donald T. Gavel y James M. Brase. *Fast wave-front reconstruction in large adaptive optics systems with use of the Fourier transforms*. J. Opt. Soc. Am. A, vol. 19, páginas 2100–2111, 2002.

[Press 06] William H. Press. *Discrete Radon transform has an exact, fast inverse and generalizes to operations other than sums along lines*. Proc. of the National Academy of Sciences USA, vol. 103, páginas 19249–19254, 2006.

[Proakis 96] JG Proakis y Dimitris G Manolakis. Digital signal processing: Principles, algorithms and applications, sec. 6.2. Prentice-Hall, third edition, 1996.

[Purcell 04] Timothy J. Purcell. *Ray Tracing on a Stream Processor.* Tesis doctoral, Stanford University, March 2004.

[Püschel 05] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan W. Singer, Jianxin Xiong, Franz Franchetti, Aca Gačić, Yevgen Voronenko, Kang Chen, Robert W. Johnson y Nick Rizzolo. *SPIRAL: Code Generation for DSP Transforms.* Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation", vol. 93, no. 2, 2005. http://www.spiral.net/index.html.

[Radon 17] J. Radon. *Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisserMannigfaltigkeiten.* Ber. Vehr. Sächs. Akad. Wiss. Leipzig. Math. Nat. Kl., vol. 69, páginas 262–277, 1917.

[Richter 80] Jean Paul Richter. The notebooks of Leonardo da Vinci. 1880. libro II, Linear perspective - The Production of pyramid of Vision, http://www.fromoldbooks.org/Richter-NotebooksOfLeonardo/section-2/item-63.html.

[Roberts 03] David E. Roberts y Trebor Smith. *The History of Integral Print Methods.* Informe técnico, Integral Resource, 2003. http://www.integralresource.org/integral_history.html.

[Roddier 91] F. Roddier y C. Roddier. *Wavefront reconstruction using iterative Fourier transforms.* Applied Optics, vol. 30, páginas 1325–1327, 1991.

[Rodríguez-Ramos 97] José Manuel Rodríguez-Ramos. *Detección de frente de onda: Aplicación a técnicas de alta resolución espacial y alineamiento de superficies ópticas segmentadas.* Tesis doctoral, Universidad de La Laguna, 1997.

[Rodríguez-Ramos 07] José M. Rodríguez-Ramos, Fernando Rosa y José G. Marichal-Hernández. *WAVEFRONT ABERRATION AND DISTANCE MEASUREMENT PHASE CAMERA; Patente WO/2007/082975; PCT/ES2007/000046*, 2007. http://www.wipo.int/pctdb/en/wo.jsp?wo=2007082975.

[Rodríguez-Ramos 08] J.M. Rodríguez-Ramos, E. Magdaleno Castelló, C. Domínguez Conde, M. Rodríguez Valido y J.G. Marichal-Hernández. *2D-FFT implementation on FPGA for wavefront phase recovery from the CAFADIS camera*. En SPIE Remote Sensing, volumen 7015, página 701539. SPIE, 2008.

[Rodríguez-Ramos 09] José M. Rodríguez-Ramos, José G. Marichal-Hernández, Fernando Rosa y Fernando Pérez Nava. *METHOD AND CAMERA FOR THE REAL-TIME ACQUISITION OF VISUAL INFORMATION FROM THREE-DIMENSIONAL SCENES, Patente WO/2009/090291; PCT/ES2009/000031*, 2009. http://www.wipo.int/pctdb/en/wo.jsp?WO=2009090291.

[Rosa 92] Fernando Rosa. *Tratamiento de imágenes astronómicas con alta resolución espacial*. Tesis doctoral, Universidad de La Laguna, 1992.

[Scharstein 02] Daniel Scharstein y Richard Szeliski. *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms*. Int. J. Comput. Vision, vol. 47, no. 1-3, páginas 7–42, 2002.

[Schechner 00] Yoav Y. Schechner y Nahum Kiryati. *Depth from Defocus vs. Stereo: How Different Really are They?* International Journal of Computer Vision, vol. 39, no. 2, páginas 141–162, 2000.

[Schiwietz 04] Thomas Schiwietz y Rüdiger Westermann. *GPU-PIV*. En Proceedings of the Vision, Modeling and Visualization Workshop VMV'04, páginas 151–158. IOS Press BV, 2004.

[Segal 03] Mark Segal y Kurt Akeley. The OpenGL graphics system: A specification. SGI, 2003. Editado por Leech, Jon, http://www.opengl.org/documentation/specs/version1.5/glspec15.pdf.

[Seiler 08] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan y Pat Hanrahan. *Larrabee: a many-core x86 architecture for visual computing*. ACM Trans. Graph., vol. 27, no. 3, páginas 1–15, August 2008.

[Shack 71] R.V. Shack y B. C. Platt. *Production and use of a lenticular Hartmann screen*. JOSA, vol. 61, no. 656, 1971.

[Stockham 66] T.G. Stockham. *High speed convolution and correlation*. En AFIPS Proceedings, volumen 28, páginas 229–233. Spring Joint Computer Conference, 1966.

[Sumanaweera 05] Thilaka Sumanaweera y Donald Liu. GPU gems 2 - programming techniques for high-performance graphics and general-purpose computation, capítulo Medical Image Reconstruction with the FFT, páginas 765–784. Addison Wesley, 2005. Editado por Pharr, Matt.

[Sutter 05] Herb Sutter. *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*. Dr. Dobb's Journal, 30(3), March 2005. http://www.gotw.ca/publications/concurrency-ddj.htm.

[Swarztrauber 87] Paul N. Swarztrauber. *Multiprocessor FFTs*. Parallel computing, vol. 5, no. 1–2, páginas 197–210, Julio 1987.

[Tarditi 06] David Tarditi, Sidd Puri y Jose Oglesby. *Accelerator: Using Data Parallelism to Program GPUs for General-Purpose Uses*. Microsoft, October 2006. http://research.

microsoft.com/research/downloads/Details/25e1bea3-142e-4694-bde5-f0d44f9d8709/Details.aspx?CategoryID.

[TM Corp. 93] TM Corp. *C* Programming Guide*. Thinking Machines Corp., Cambridge, Massachusetts, May 1993. http://bradley.csail.mit.edu/cm5docs/CStarProgrammingGuide.pdf.

[Toft 96] Peter Toft. *The Radon Transform - Theory and Implementation*. Tesis doctoral, Department of Mathematical Modelling, Technical University of Denmark, June 1996.

[Tyson 98] Robert K. Tyson. Principles of adaptive optics. Academic Press, 2 edition, Jan. 1998.

[van Loan 92] Charles van Loan. Computational frameworks for the fast fourier transform. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[Veeraraghavan 07] Ashok Veeraraghavan, Ramesh Raskar, Amit Agrawal, Ankit Mohan y Jack Tumblin. *Dappled photography: mask enhanced cameras for heterodyned light fields and coded aperture refocusing*. ACM Trans. Graph. (SIGGRAPH 07), vol. 26, no. 3, página 69, 2007. http://www.umiacs.umd.edu/~aagrawal/sig07/index.html.

[Veeraraghavan 08] Ashok Veeraraghavan, Amit Agrawal y Ramesh Raskar. *Non-Refractive Modulators for Encoding and Capturing Scene Appearance and Depth*. Informe técnico TR2008-028, Mitsubishi Electrics Research Laboratory, July 2008. "http://www.merl.com, published on CVPR 08".

[Viola 04] Ivan Viola, Armin Kanitsar y Meister Eduard Gröller. *GPU-based Frequency Domain Volume Rendering*. En Proceedings of SCCG 2004, páginas 49–58, Apr 2004.

[Wilburn 02] Bennett Wilburn, Michael Smulski, Hsiao-Heng Kelin Lee y Mark Horowitz. *The Light*

Field Video Camera. En Proceedings of Media Processors 2002, SPIE Electronic Imaging 2002. SPIE, 2002.

[Wilburn 05] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz y Marc Levoy. *High Performance Imaging Using Large Camera Arrays.* ACM Transactions on Graphics, vol. 24, no. 3, páginas 765–776, July 2005. Proceedings of ACM SIGGRAPH 2005.

[Wloka 03] Matthias M. Wloka. *Implementing a GPU-Efficient FFT*, 2003. SIGGRAPH course slides.

[Wu 98] Tung-Kuang Wu y Martin L. Brady. *A Fast approximation algorithm for 3D image reconstruction.* En Workshop on Image Processing and Character Recognition, páginas 213–220, 1998.

[Yu 04] Jingyi Yu, Leonard McMillan y Steven J. Gortler. *Surface Camera (Scam) Light Field Rendering.* International Journal of Image and Graphics, vol. 4, no. 4, páginas 605–625, 2004.

[Zhang 03a] Cha Zhang y Tsuhan Chen. *On generalized sampling for image-based rendering data.* Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on, vol. 3, páginas III–469–72 vol.3, April 2003.

[Zhang 03b] Cha Zhang y Tsuhan Chen. *Spectral analysis for sampling image-based rendering data.* Circuits and Systems for Video Technology, IEEE Transactions on, vol. 13, no. 11, páginas 1038–1050, Nov. 2003.

[Zhang 04] Cha Zhang. *On sampling of image-based rendering data.* Tesis doctoral, Carnegie Mellon University, Pittsburgh, PA 15213, June 2004.