Proyecto Final de Carrera

Ingeniería en Informática

Multi-view image rendering for holographic stereogram printing

David Ángel Valle Badenas

Enero 2013

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Director: **Atanas Gotchev**

TAMPERE UNIVERSITY OF TECHNOLOGY

Ponente: **Francisco José Serón**

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

RESUMEN

Desde que el ser humano ha sido capaz de reproducir objetos del mundo real en material sintético, el objetivo ha estado determinado por concebir sistemas de experiencias virtuales más precisos, desde las primeras fotografías estereográficas hasta las contemporáneas producciones cinematográficas en 3D. Los productos actualmente disponibles en el mercado son consecuencia de investigaciones iniciadas décadas atrás. Sin embargo, nuevas exploraciones en tecnologías más complejas han dado lugar a nuevas inmersiones de realidad virtual. Los estereogramas holográficos, los cuales proveen una perspectiva más amplia y una mayor cantidad de puntos de vista, se suponen ser el siguiente paso en la evolución de los sistemas visuales 3D. Esta tecnología no ha sido introducida en el mercado todavía, sin embargo, el progreso realizado en el campo del hardware que permite una mayor cantidad de almacenamiento y proceso de datos y, mejores diseños y algoritmos de software que consiguen resultados de forma más eficiente, harán posible la disponibilidad de pantallas holográficas al consumidor en un futuro cercano.

Actualmente, los objetos 3D pueden ser manipulados fácilmente y la posibilidad de obtener su información digital lleva al desarrollo de hologramas generados por ordenador basados en sistemas multi-vista. La captura y post-procesado de las vistas necesarias para crear un holograma generado por ordenador conlleva un coste de almacenamiento y computación alto. Esta tesis afronta el problema de optimización del número de vistas o cámaras requeridas para reconstruir una escena compuesta por un número de vistas totalmente densa. El método desarrollado parte de una escena 3D y un set completo de vistas. Solo algunas de esas vistas son utilizadas para reproducir la escena completa. A través del análisis de las imágenes en el dominio espacial y de frecuencia de Fourier, se han utilizado técnicas de procesado de señal para interpolar y regenerar la escena original. Se han considerado un número diferente de vistas disponibles inicialmente, las cuales son procesadas para reconstruir la escena *a posteriori*, de manera que se pueda establecer un ratio entre el número mínimo de cámaras necesarias y el número total de cámaras que la escena contiene. Los resultados de este estudio sugieren, comparando la semejanza entre las escenas originales y sus versiones optimizadas, que la reconstrucción de todas las vistas puede ser obtenida a partir de muchas menos cámaras y sin una pérdida de calidad considerable. El siguiente texto muestra el análisis del problema y las técnicas utilizadas para resolverlo. La memoria concluye con una valoración de las técnicas y herramientas desarrolladas, así como de los resultados obtenidos.

AGRADECIMIENTOS

Este proyecto ha sido realizado en la Tampereen Teknillinen Yliopisto (Tampere University of Technology) en el departamento de Signal Processing. El proyecto supone un Proyecto Final de Carrera por la Universidad de Zaragoza, dentro del programa Erasmus.

Quiero dar las gracias a mi supervisor de proyecto Dr. Atanas Gotchev por toda su dedicación y paciencia, especialmente en aquellos momentos en los que mis conocimientos previos no eran suficientes para entender algunos temas. También quiero agradecer al profesor Dr. Robert Bregovic por su tiempo dedicado a resolver mis dudas.

Quiero agradecer a mi supervisor en España, Dr. Francisco J. Serón, por su apoyo y tiempo, especialmente cuando sólo me era posible estar en España por un tiempo limitado.

Finalmente, quiero dar las gracias a mi familia, amigos y personas especiales que me han apoyado a lo largo de todo el proyecto.

The scientist is not a person who gives
the right answers,
he is one who asks the right questions.

Claude Lévi-Strauss

Tabla de contenidos

1 Introducción **1**

1.1 Contexto y motivación 1

1.2 Alcance y objetivo del proyecto 6

1.3 Estructura 6

2 Conceptos teóricos **7**

2.1 Método de holograma generado por ordenador 7

2.2 Representación de EPI 8

2.3 Requisitos de la propuesta 10

3 Análisis **13**

3.1 Relaciones entre las características de la escena y la representación en el dominio de Fourier 13

3.2 Análisis de la transformación de dominio de las imágenes EPI 14

3.3 Filtrado de EPI y optimización del número de vistas 17

4 Diseño del sistema multi-vista **20**

4.1 Construyendo el sistema multi-vista para escenas 3D 21

4.2 Creación y pre-filtrado de EPIs 23

4.3 Optimización del número de vistas 25

5 Resultados **29**

5.1 Algoritmo Fast DCT 29

5.2 Resultados experimentales 30

6 Conclusiones **34**

6.1 Conclusiones sobre los resultados 34

6.2 Trabajo futuro 36

6.3 Valoración personal 36

7 Apéndices **41**

7.1 Original MSc Thesis at Tampere University of Technology 41

7.2 Metodología de trabajo 85

7.3 Diagrama de Gantt 85

7.4 Herramientas utilizadas 85

7.5 Código - Sistema de rendering multi-vista 86

7.6 Código - Recortado de renders 92

7.7 Código - Creación de EPIs con densidad completa 93

7.8 Código - Pre-filtrado de EPIs 94

7.9 Código - Interpolación de EPIs 96

7.10 Código - Reconstrucción de vistas 99

7.11 Código - Algoritmos Fast DCT 100

7.12 Código - Tests Fast DCT 104

7.13 Código - Tests 108

7.14 Gráficas de los tests 114

Índice de figuras

1.1 Holograma en Star Wars 2
1.2 Horizontal Parallax Only vs. Full-parallax stereograms 3
1.3 Sección cruzada de una cámara con el plano de proyección 3
1.4 Disposición de la mesa holográfica. 4
1.5 Lentes de colimación 4
1.6 Esquema de grabación holográfica típica 5
1.7 Ejemplo de reconstrucción holográfica 5

2.1 Representación del hogel (hologram element) 8
2.2 Disposición de cámaras para full o horizontal parallax-only 9
2.3 Proceso de construcción de la EPI 10
2.4 Ejemplo de oclusión en la EPI 10
2.5 Cuatro posibles configuraciones de disposición de cámaras. 11
2.6 EPIs obtenidas con diferentes configuraciones de cámaras 12

3.1 Función de profundidad representada en el dominio espacial y epipolar . . 14
3.2 Soporte espectral del campo de luz limitado en el dominio de frecuencia. . 14
3.3 Representación de EPI y la transformada de Fourier en 2D. 15
3.4 Pasos del anti-aliasing para una señal unidimensional. 16
3.5 Comparación entre EPIs filtradas con anti-aliasing y no filtradas. 17

4.1 Esquema del proceso de desarrollo 20

4.2 Interfaces en Blender para los scripts paralelo y convergente 21
4.3 Ejemplo 3D de cámaras paralelas y convergentes en Blender. 21
4.4 Problema de recortado de vistas con cámaras paralelas 22
4.5 Solución para el recortado de renders. 23
4.6 Muestras de vistas. 23
4.7 Muestras de EPIs. 24
4.8 Muestras de EPIs filtradas con anti-aliasing. 25
4.9 Población de matrices con kron, repmat y ones. 27
4.10 Perspectiva gráfica del proceso completo. 28

5.1 Muestra de señales escaladas con el algoritmo Fast DCT. 30
5.2 Comparación de escalado iterativo entre los métodos bilinear, bicubic y Fast DCT. 30
5.3 Efectos en los bordes en el método DCT 31
5.4 Comparación de tiempos entre Fast DCT y el resto de kernels de interpolación. 31
5.5 PSNR de las vistas reconstruidas para cada método de interpolación. . . . 32
5.6 PSNR promedio para cada factor en comparación con vistas basadas en aliased EPIs. 33

6.1 Promedio de PSNR para cada factor y rango de valores PSNR aceptables. 35

7.1 Diagrama de Gantt 85
7.2 PSNR de vistas reconstruidas con cada método de interpolación, factor 2 . 114
7.3 PSNR de vistas reconstruidas con cada método de interpolación, factor 4 . 114
7.4 PSNR de vistas reconstruidas con cada método de interpolación, factor 8 . 115
7.5 PSNR de vistas reconstruidas con cada método de interpolación, factor 16 115
7.6 PSNR de vistas reconstruidas con cada método de interpolación, factor 32 116

1. Introducción

En este trabajo revisamos el método de impresión de estereogramas holográficos estudiado, y su implementación a través de un sistema de rendering multi-vista. En este capítulo se presentan los conceptos fundamentales de impresión holográfica, así como la motivación encontrada para explorar este tema, y la estructura del resto del documento.

1.1 Contexto y motivación

Desde las primeras pantallas estereoscópicas, los investigadores han intentado mejorar la experiencia de realidad virtual llevando a los consumidores entornos 3D más realísticos e inmersivos. El reto se halla en capturar la escena desde el mundo real y mostrar la información capturada en un material sintético. Algunas veces, precisamente el caso trabajado en este estudio, el material usado como fuente no pertenece al mundo real; es generado por ordenador, y por lo tanto, más fácil de manipular.

Las primeras pantallas holográficas, desarrolladas en 194 por Leith [1], estaban iluminadas por láser. Desde entonces, la prensa y producciones cinematográficas mostraron hologramas como una manera futurística de grabar y mostrar el mundo en tiempo real. El mejor ejemplo de todo, el primer largometraje de Star Wars en 1977 (un año más tarde del desarrollo de Leith) ya introdujo holocámaras y holoproyectores como muestra la Fig. 1.1.

Un holograma contiene el patrón de interferencia producido por las longitud de onda de referencia y del objeto, que poseen diferente fase. A diferencia de las típicas fotografías estereoscópicas, un holograma es el medio de grabación y la pantalla al mismo tiempo. Los hologramas tuvieron la desventaja de algunos requisitos como por ejemplo, el tener que ser capturados utilizando una luz monocromática y coherente. Además, los objetos capturados tenían que permanecer estables dentro de una fracción de longitud de onda de la luz [2].

A diferencia de como las recientes pantallas sintéticas muestran hologramas [9], los hologramas tradicionales requerían de un almacenamiento masivo de información, haciendo la computación un proceso pesado. Las estereografías holográficas suponen que sólo un número limitado de puntos de vista son grabados, según la percepción humana

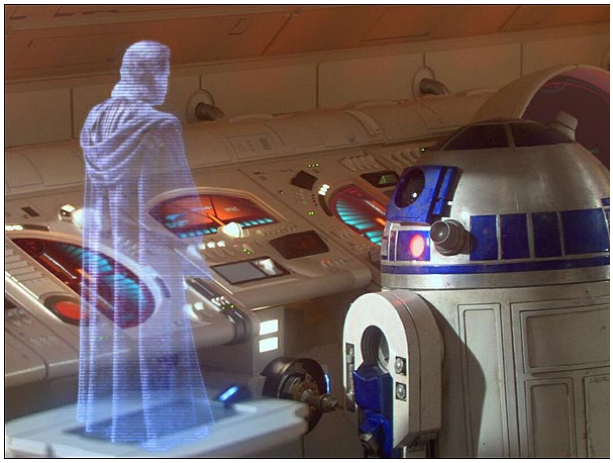


Figura 1.1: Fotograma de Star Wards donde un holograma es proyectado desde abajo en una mesa.

deseada más que la capacidad de almacenamiento del holograma. Esto supone una diferencia notable respecto el primer holograma. Otra gran diferencia establece un proceso de captura por separado. Como cualquier estereograma tradicional, las imágenes pueden ser capturadas con una cámara fotográfica ordinaria o a través de técnicas gráficas por ordenador más adelante. Además, se puede sustituir el objeto real por un modelo generado por ordenador, que puede ser configurado virtualmente sin coste alguno.

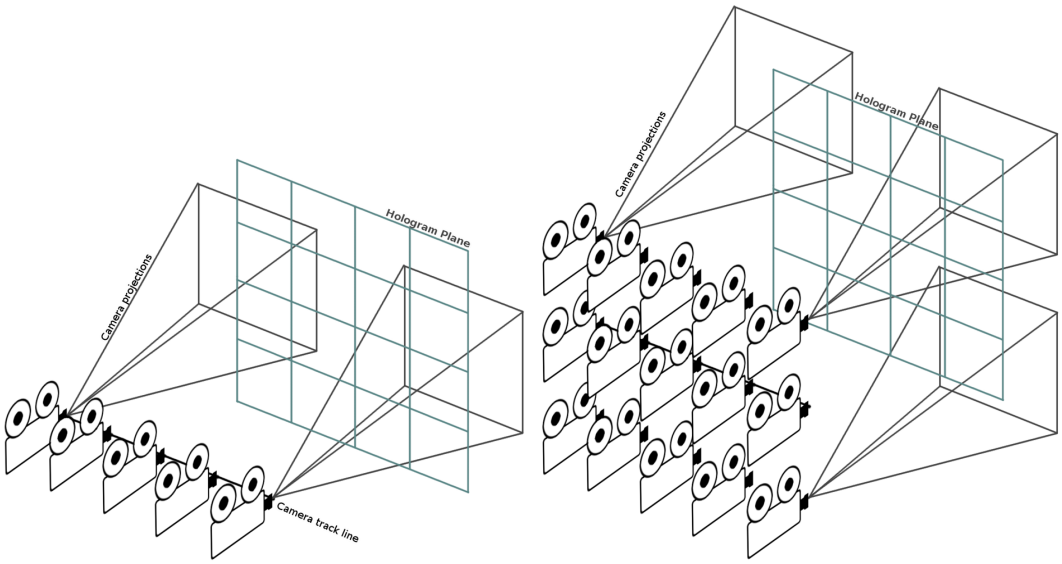
La grabación de un estereograma es dada en tres fases [2]:

1. Captura fotográfica, bien con objetos reales o modelos sintéticos en 3D.
2. Grabación holográfica.
3. Vista final.

Antes de avanzar más en profundidad en el proceso holográfico de grabación, algunos términos tienen que ser explicados para entender los distintos métodos, en referencia a la adquisición de vistas de un objeto como son los estereogramas Full-parallax y Horizontal Parallax Only (HPO). Mientras el primero intenta simular el mundo real, el segundo ofrece una visión 3D decente con muchas menos cámaras. Fig. 1.2 ilustra esta diferencia.

Todas las cámaras apuntan hacia delante, y sus ejes son paralelos al plano de holograma y la pista de cámaras. Las proyecciones de las cámaras, con forma de pirámide, representan sus campos de visión. Dichas pirámides son intersectadas por el plano de holograma, resultando en un plano de proyección para cada cámara. Una vista más detallada de la sección resultado de la pirámide es mostrada en la Fig. 1.3. Se usa una configuración de cámaras paralelas para explicar cómo un sistema multi-vista funciona, sin embargo, esta configuración será discutida más tarde en siguientes capítulos.

Una placa holográfica está compuesta por hendiduras verticales a lo largo de ella. Cada hendidura está expuesta a una imagen proyectada en una pantalla trasera, representando



(a) (b)

Figura 1.2: a) Esquema de un estereograma of. Las cámaras son colocadas a lo largo de la pista, paralela al plano de holograma. b) Esquema de un estereograma full-parallax. Las cámaras son colocadas a lo largo de un plano paralelo al plano de holograma.

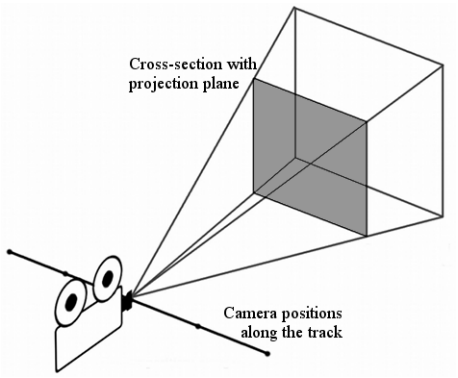


Figura 1.3: En esta configuración, las cámaras siempre apuntan hacia delante. Se puede observar que la sección cruzada entre la pirámide y el plano de producción se mueve acorde con la pista del estereograma.

diferentes puntos de vista. Una vez el holograma es grabado, cada punto de vista puede ser visto a través de cada hendidura. Un vidente mirando al estereograma verá dos aperturas, una para cada ojo. El cerebro interpreta las diferencias entre las dos vistas como información tridimensional. La apariencia de una escena 3D funciona de acuerdo a un indicativo de profundidad binocular. Los ojos del vidente siguen dos diferentes hendiduras en el holograma. Las imágenes que aparecen en esas hendiduras coinciden directamente enfrente de los ojos, produciendo un efecto estereoscópico de estar en el infinito. Sin embargo, el observador enfoca su vista en el plano de proyecto, situando ahí la imagen. Si el vidente se mueve lateralmente, un nuevo par de imágenes son vistas,

simulando la escena 3D. Como las hendiduras son grabadas horizontalmente, siguiendo el esquema HPO, el vidente ve siempre la misma perspectiva vertical independientemente de la posición vertical desde donde se observe. De hecho, la imagen se moverá como si estuviera en el plano de proyección vertical.

De acuerdo a la grabación holográfica presentada en [2], se utilizan dos haces de luz coherente. El haz de láser es dividido en dos, originando un haz de referencia y uno del objeto. Este haz del objeto es ampliado cuando pasa a través de la cámara y las lentes de proyección. Ilumina el holograma proyectando en la pantalla de proyección una imagen grabada en film de cine en la cámara. Los fotogramas de proyección representan un objeto 2D localizado en un plano de proyección sin límites, y al mismo tiempo la placa con el mecanismo de hendiduras están encarados uno al otro, como se puede ver en la Fig. 1.4 con más precisión. La placa holográfica esta expuesta a la imagen proyectada y el haz de referencia. Sin embargo, sólo una hendidura del holograma está expuesta, con un pieza opaca perforada con agujeros cubriendo la placa del holograma.

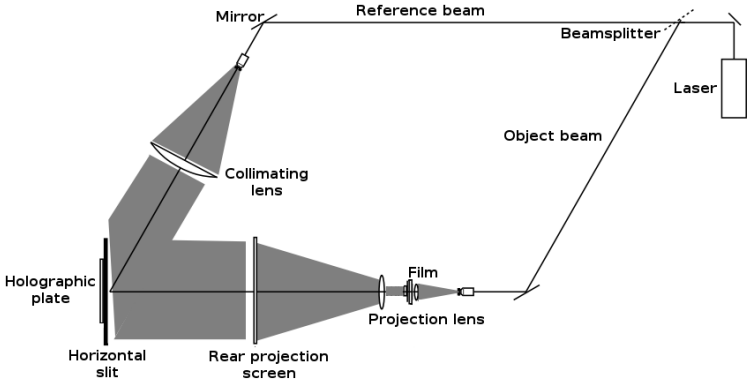


Figura 1.4: Disposibión de la mesa holográfica vista desde arriba (Fuente [2]).

Cada hendidura tiene que ser iluminada con el haz de referencia en la misma dirección, independientemente de la posición lateral de la hendidura. Esto se consigue juntando el haz con una lente de colimación o Fresnel, el mismo principio usado en los faros. Fig. 1.5 da una idea más detallada de cómo un colimador funciona.

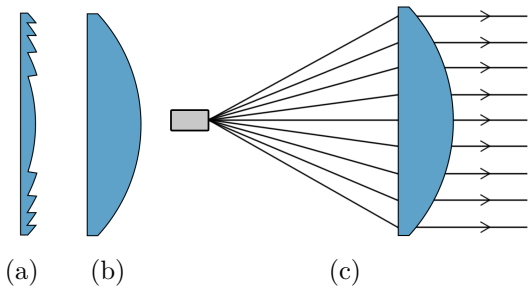
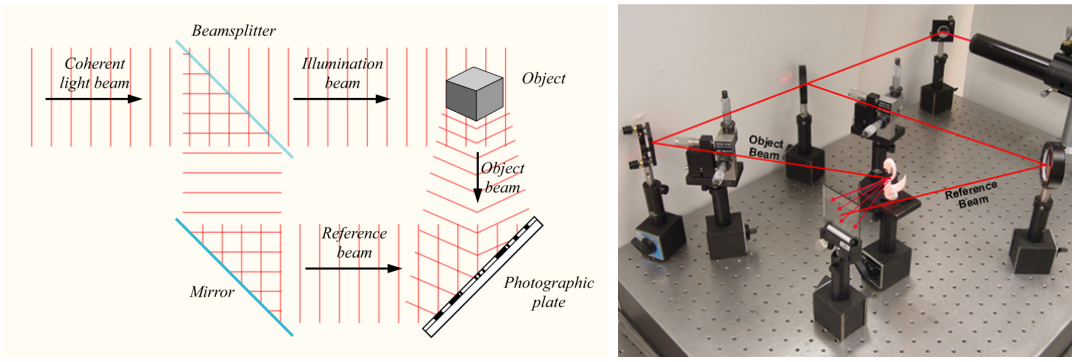


Figura 1.5: a) Lente Fresnel usada en faros. b) Lente plano-convexa equivalente. c) Esquema de un colimador que alinea los haces produciendo una salida paralela.

Otra configuración típica usada en grabación holográfica consiste en lanzar dos haces

de láser a un objeto físico y grabar la interferencias de longitud de onda que el objeto desprende junto con el haz de referencia. En la Fig. 1.6 se detalla el esquema. En vez de lentes de colimación, se usan lentes y espejos para redirigir los haces hacia la placa holográfica. La grabación es realizada en la oscuridad para evitar cualquier tipo de interferencia con la luz natural.



(a) (b)

Figura 1.6: a) Esquema de grabación holográfico con un objeto expuesto a un haz de láseres. *Bob Mellish.* b) Fotografía del mismo escenario en un laboratorio. *College of Optics & Photonics, University of Central Florida.*

Cuando una placa holográfica es iluminada por un haz de referencia equivalente, se puede ver una reconstrucción del objeto. Si la posición de la placa o el observador cambia, la apariencia del objeto cambia también como si fuera un objeto real. Un ejemplo de reconstrucción holográfica se muestra en la Fig. 1.7.

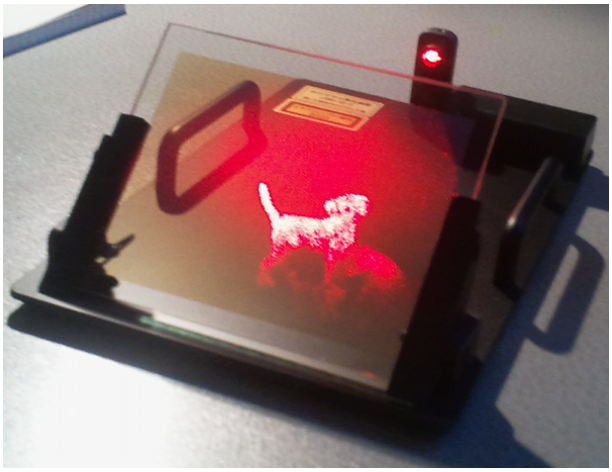


Figura 1.7: El perro es reconstruído aplicando un haz láser de referencia al holograma. *Holograma presentado por el Dr.Kenji Yamamoto en la 3D Media Training School, Tampere, 2012.*

1.2 Alcance y objetivo del proyecto

El objetivo de este proyecto es entender el proceso de creación de hologramas para desarrollar nuevas técnicas. Se desarolla el software necesario con el propósito de proveer la información necesaria para los siguientes etapas del proceso.

Los hitos a alcanzar en este proyecto incluyen:

- Diseño y desarrollo de un script que provee un set de imágenes renderizadas correspondientes a un sistema horizontal-parallax only desde una escena 3D.
- Creación de imágenes epipolares (EPI) a partir de las imágenes renderizadas, permitiendo representar todas las vistas en una imagen.
- Análisis de las imágenes EPI en el dominio de la frecuencia. Relación entre características de la escena y su representación en el dominio de Fourier.
- Diseño y desarrollo de un filtro para procesas las imágenes EPI, optimizando el número mínimo de cámaras.

1.3 Estructura

Este documento está organizado con la siguiente estructura:

El capítulo 2 describe el estado del arte relacionado con el tema.

El capítulo 3 presenta el análisis teórico que es combinado con los experimentos.

El capítulo 4 explica todas las herramientas desarrolladas para llevar a cabo los experimentos y justificar su propósito.

El capítulo 5 muestra los resultados obtenidos y su representación gráfica.

El capítulo 6 completa la investigación proveyendo las conclusiones y el trabajo futuro que puede proseguir el presente estudio.

El capítulo 7 contiene las referencias bibliográficas usadas para el desarrollo de este estudio.

Los capítulos siguientes contienen los apéndices incluyendo el resto del desarrollo completado durante este trabajo. La tesis desarrollada originalmente en inglés conformará el primer anexo.

2. Conceptos teóricos

Este capítulo revisa el estado del arte de trabajos anteriores así como todo el conocimiento necesario para entender la investigación realizada. También revisa la propuesta de impresión de estereogramas holográficos.

2.1 Método de holograma generado por ordenador

Como se ha explicado en el capítulo anterior, un holograma contiene información del campo de luz. Un holograma estereográfico es una versión discretizada compuesta de varios elementos de holograma llamados hogel (hologram element). Cada hogel contiene información 3D desde varias perspectivas, la cual ha sido previamente grabada dependiendo del ángulo de refracción (i.e: el punto de vista del observador).

Un holograma completamente computado ofrece un infinito número de perspectivas. Por otro lado, un estereograma holográfico ofrece sólo un número finito de perspectivas, usando una reconstrucción por longitud de onda. Los estereogramas holográficos forman una aproximación de la longitud de onda decrementando el tiempo de computación. Esta aproximación es hecha con un número discreto de perspectivas. La propuesta desarrollada en este estudio persigue la optimización del número de vistas necesarias [6], obteniendo una percepción holográfica similar.

Un método de codificación relacionado con la difracción [8] hace referencia al estereograma holográfico como la suma de las amplitudes moduladas de la señal chirp. Una señal chirp incrementa su frecuencia con el tiempo. Su amplitud modulada en el plano del holograma produce un set de emisores direccionales en el plano emisor. Cada chirp enfoca la luz para crear un punto emisor, mientras la luz de las vistas dependiente del ángulo es codificada en su modulación de amplitud.

De acuerdo a Quinn [8], la computación paralela vectorial de un holograma está dividida en tres pasos:

1. Pre-computación de los vectores chirp en texturas chirp.

2. Rendering multi-vista de la escena directamente en los vectores de modulación alma-

cenados en la textura de modulación usando una cámara de doble cono (capturando vista delantera y trasera).

3. Montaje del holograma juntando los vectores de modulación y chirp mediante la obtención de la textura y ejecutando un producto escalar.

Este trabajo se enfocará en una porción que es desarrollada durante el segundo paso, con algunas variaciones, e.g: horizontal parallax-only en vez de usar cámaras con doble captura. Las diferencias exactas se discutirán en Sección 2.3 al final de este capítulo, donde se presentan los requisitos del método desarrollado.

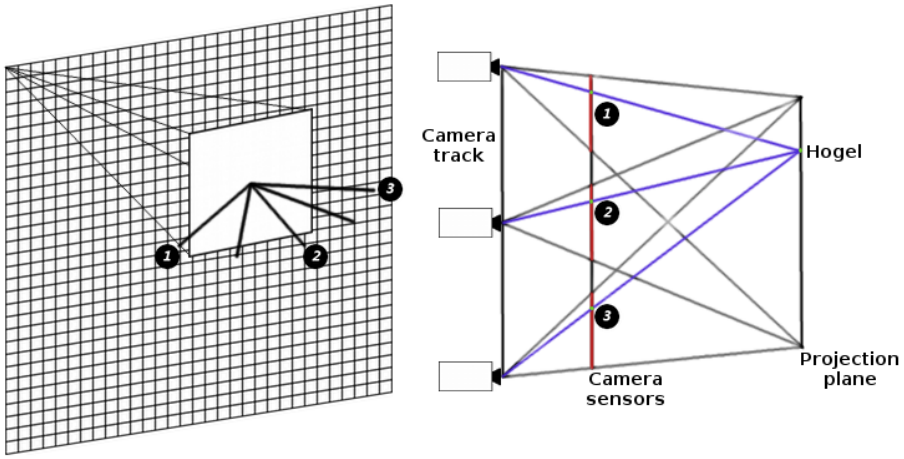


Figura 2.1: A la izquierda, el holograma y una imagen aumentada de un hogel. A la derecha, un píxel almacenado en el sensor de la cámara, representando el mismo píxel de la escena, es grabado en el hogel. Los sensores de la cámara (en color rojo) están situados en realidad en la cámara.

Un método más general de entender cómo un sistema de rendering multi-vista funciona esta basado en los hogels introducidos previamente. Fig. 2.1 mostrará inteligiblemente la siguiente explicación. Cada hogel contiene la onda de luz correspondiente a un píxel en la escena, desde cada punto de vista. Esta información puede ser reconstruída aplicando la luz apropiada al hogel, que emitirá la información grabada dependiendo del punto de vista. Varias cámaras graban desde diferentes posiciones a lo de la pista de cámaras. Cada sensor está compuesto por muchos píxeles. Un solo hogel contiene la información asociada al mismo píxel en cada vista. Esto significa que contiene información de tantas direcciones diferentes como número de vistas usadas en la escena.

2.2 Representación de EPI

La correlación entre imágenes desde diferentes posiciones en la misma escena estática connota la coherencia de perspectiva. La principal distinción entre la coherencia de perspectiva y coherencia temporal es que esta última es que la primera es más restrictiva

que la segunda, en relación a los cambios en la imagen a lo largo del tiempo. Las características de la escena, i.e: cambios en el sombreado y geometría, pueden ser analizados en el dominio epipolar.

Una Epipolar Plane Image (EPI) es una representación 2D del campo de luz. Constituye la transformación de un set de imágenes multi-vista en un corte multi-perspectiva del campo de luz. Para obtener esta representación, primero las cámaras tienen que ser dispuestas como se muestra en Fig. 2.2. Halle [5] describe estas dos diferentes disposiciones como cámaras regular shearing (RS): planar regular shearing (PSR) y linear regular shearing (LSR).

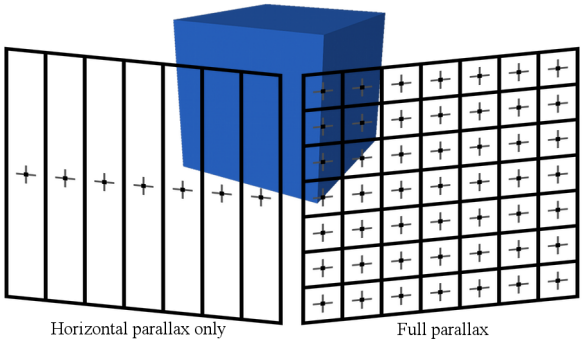


Figura 2.2: A la izquierda, la disposición LSR provee horizontal parallax-only mientras que a la derecha, PSR, captura información horizontal y vertical.

La disposición LSR, la cual consiste de un array de cámaras representa una configuración HPO, proveyendo toda la información necesaria con una sola fila de cámaras, pero removiendo el parallax vertical. El vidente recibirá la misma imagen independientemente de posición vertical que tome. Por otra parte, la disposición PSR provee full parallax, pero requiere muchas más cámaras.

Comprender cómo se forman las EPIs ayuda a entender el sistema de impresión holográfico, dado que las EPIs conforman un elemento esencial en el proceso. Imagina que todas las imágenes renderizadas del sistema multi-vista son apiladas como una baraja de cartas. La imagen correspondiente a la vista más a la derecha se coloca al frente, y la más situada a la izquierda, atrás. A A continuación, el volumen espacio-perspectiva creado se filetea horizontalmente. El corte multi-perspectiva obtenido, i.e: la EPI, contiene la misma fila equivalente de cada vista y está tumbada, mirando hacia arriba. En otras palabras, la última fila de la EPI pertenece a la vista más a la derecha y la primera fila pertenece a la de más a la izquierda. Fig. 2.3 muestra gráficamente la técnica usada para obtener la EPI.

Aunque Fig. 2.3 muestra una sola EPI, todas ellas deberían ser calculadas para construir el volumen espacio-perspectiva completo. Existen tantas EPIs como la resolución vertical de las vistas.

La conveniencia de usar EPIs es fuertemente notable en relación con las posibilidades de usar algunas características en futuros desarrollos. Por ejemplo, su estructura linear

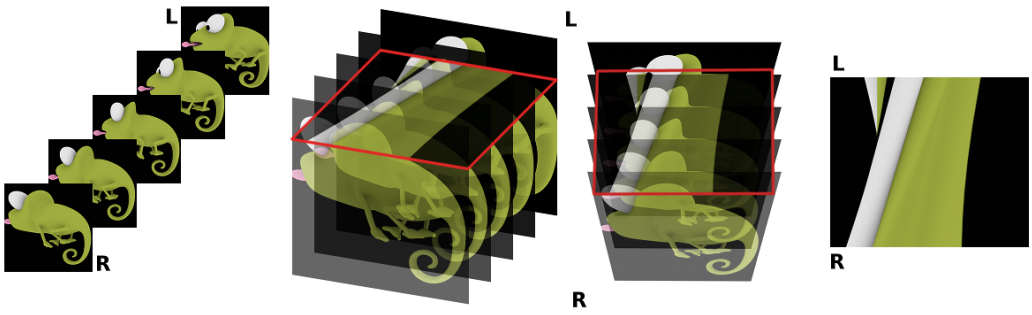


Figura 2.3: A la izquierda, 5 vistas. En el medio, ambas perspectivas 3D muestran cómo las vistas son apiladas. El marco rojo designa la fila elegida para esta EPI, que reposa horizontalmente. A la derecha, la EPI resultante contiene la vista más a la izquierda arriba, y la vista más a la derecha abajo. Nótese cómo el ojo derecho del camaleón desaparece de la EPI según la cámara se mueve a la derecha.

permite aplicar efectivamente algoritmos de interpolación. Esto es totalmente importante porque uno de los objetivos de este proyecto (explicado en más detalle en el capítulo siguiente) responde a la optimización del número de vistas. Adicionalmente, las EPIs reúnen características extra, como detección de caras, sombreado y oclusión de objetos. Este último puede ser observado en Fig. 2.4 donde se ha seleccionado una fila diferente.

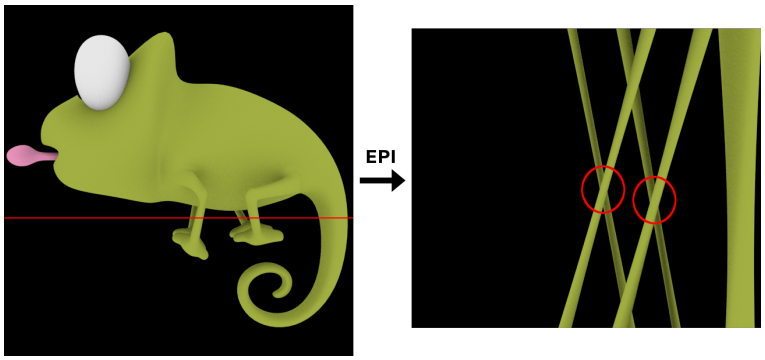


Figura 2.4: A la izquierda, el camaleón 3D y la fila elegida marcada en rojo. A la derecha, la EPI resultante donde las oclusiones producidas por ambas piernas están encerradas en un círculo rojo.

2.3 Requisitos de la propuesta

Después de presentar los conceptos teóricos necesarios, este capítulo describirá los requisitos adoptados por la propuesta realizada y las razones de por qué ciertos parámetros son elegidos y por qué otros son descartados.

Antes de todo, como el sistema de impresión holográfico está basado en imágenes multi-vista, se tiene que elegir una configuración determinada con referencia a las cámaras:

especifica su posición y rotación con relación a la escena. Hay disponibles cuatro configuraciones: cámaras apuntando al frente paralelamente al plano de proyección y tres tipos de cámaras recentradas; dos de ellas están automáticamente descartadas por las distorsiones que introducen. Fig. 2.5 muestra de una manera más inteligible la disposición de las cámaras en estas cuatro configuraciones. El tamaño del plano de proyección y su distancia a la linea de cámaras determinará el ángulo de visión y distancia focal especificada en la lente de la cámara. Además, el plano de proyección conforma la sección cruzada con la pirámide proyectada desde la cámara, haciendo capaz a la cámara de capturar apropiadamente el área correspondiente.

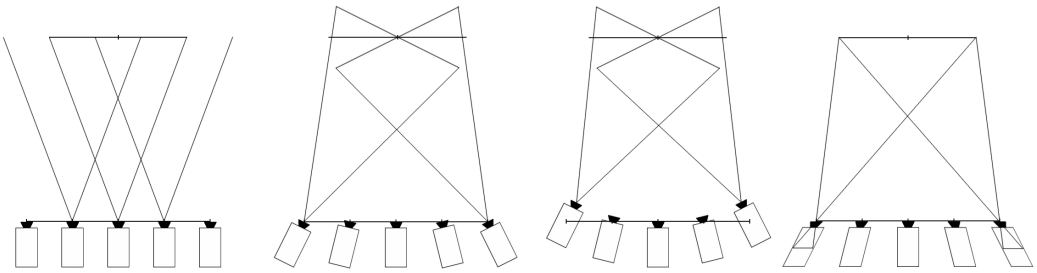


Figura 2.5: A la izquierda, cámaras paralelas. En el medio, cámaras convergentes enfocan al centro de la escena a lo largo de la pista y giran alrededor del centro de la escena. Estas dos configuraciones conllevan a distorsiones. A la derecha, cámaras recentradas con los sensores desplazados lateralmente de acuerdo con la lente para mantener la imagen centrada.

Por lo tanto, la segunda y la tercera configuración deben ser descartadas. Violan las correspondencias en términos de geometría entre la posición de las cámaras y la posición del observador. El lugar en el área donde las imágenes son tomadas debe corresponder con el punto de vista del estereograma. La pista a lo largo de la cual se mueven las cámaras, debe ser recta, manteniendo la misma geometría que la placa holográfica plana. La cámara toma una vista desde cada posición correspondiente a la posición de las hendiduras. Además, cada punto debe permanecer en el mismo lugar en la escena independientemente desde dónde es capturado el punto de vista. De esta manera, la profundidad se mantiene en cada vista. Escenas complejas con puntos muy cercanos y distantes deberían conservar la misma profundidad para cada punto de vista. Estas dos condiciones hacen que las dos configuraciones mencionadas arriban sean incompatibles. Además, la EPI obtenida, por ejemplo, con la segunda configuración, carece de propiedades que la hagan útil, como líneas rectas.

Con respecto a las cámaras paralelas o recentradas, las primeras tienen una desventaja. La mayoría de los puntos de vista contienen información que no puede ser usada. Por lo tanto, es información inútil que es procesada pero nunca grabada en el holograma final. Las cámaras paralelas conllevan una configuración más sencilla, pero su desventaja implica demasiada información fuera del plano de proyección. La cuarta configuración mostrada en Fig. 2.5 es la elegida. Una manera de conseguir esas imágenes renderizadas, como es explicado más en profundidad en el siguiente capítulo, consiste en usar la primera configuración y cámaras con un campo de visión que abarque el plano de proyección incluso

(a) (b) (c)

Figura 2.6: a) Las cámaras paralelas preservan las correspondencias geométricas, pero la mayor parte, en negro, suponen información inútil. b) La EPI presenta geometría deformada causada por una configuración errónea. c) Las cámaras recentradas proveen información útil independientemente del punto de vista y preservan las correspondencias geométricas.

desde el punto de vista más extremo (lateralmente hablando). Fig. 2.6 presenta las EPIs finales que serían producidas por las diversas configuraciones de cámara.

Los parámetros elegidos para el siguiente desarrollo son los siguientes. La resolución de la cámara es de 600 píxeles y 3 capas dado el espacio de color RGB. El número de vistas elegidas es 512, lo que simplificará post-procesados de los renders al trabajar con potencias de 2. Renderizar 512 vistas de una escena compleja puede resultar en un proceso de computación largo y pesado. Optimizar el número de cámaras a través de interpolación será una de las tareas a ser pulidas.

3. Análisis

Como ha sido expuesto en capítulos anteriores, optimizar el número de cámaras implica el análisis de interpolación y re-muestreado. Con este propósito, el análisis de EPIs tiene que hacerse en el dominio espacial y de frecuencia para entender como manipular las imágenes.

3.1 Relaciones entre las características de la escena y la representación en el dominio de Fourier

Antes de afrontar el estudio del filtrado e interpolación de imágenes epipolares, es apropiado tener un primer contacto con las propiedades de la escena y su representación en el dominio de Fourier, i.e: la profundidad. La profundidad juega un papel muy importante en estereogramas holográficos. Como diferentes profundidades afectan la representación de objetos en el dominio epipolar y de frecuencia se explicará en esta sección.

Inicialmente, se considera un punto *z(v,t)* en una escena en particular (Fig. 3.1). El punto es visto por dos cámaras, *c1* y *c2* con una distancia focal *f*, las cuales se mueven a lo largo de la pista de cámara *t*, grabando la posición horizontal donde caerían los dos puntos (el mismo punto visto desde dos posiciones diferentes). Estas dos posiciones donde el punto *z(v,t)* es visto desde las cámaras son *v* para *c1* y *v'* para *c2*. La función de disparidad $v - v' = ft/z$ describe la geometría de la escena.

Fig. 3.2 muestra el modelo de una señal con profundidad constante. Las líneas rojas representan la profundidad mínima y máxima. También se observa el soporte espectral del campo de luz. La línea azul, con un ángulo de 45º respecto de los ejes representa la profundidad constante. Las líneas rojas determinan los límites de de profundidad del soporte espectral. La inclinación considerada en este dominio puede ser traducida como la profundidad de los objetos en la escena.

De acuerdo a Chai[6], "Cualquier escena con una profundidad entre z_{min} y z_{max} tendrá su soporte espectral continuo limitado en el dominio de frecuencia". El uso de imágenes epipolares conecta las características de la escena, como la profundidad, con el dominio de Fourier, el cual puede ser adecuadamente manipulado. Finalmente, un par de ejemplo

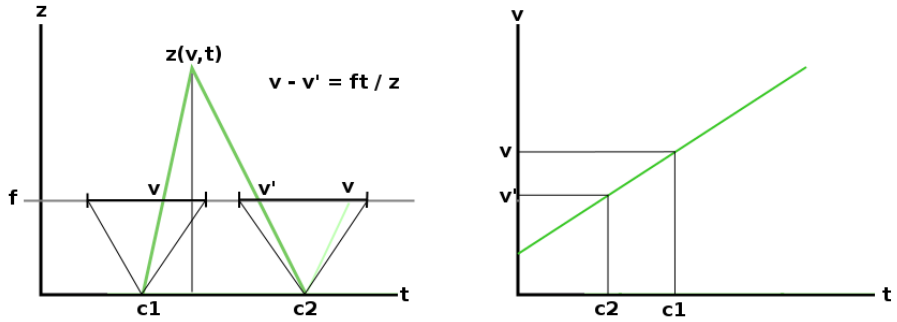


Figura 3.1: A la izquierda, el punto situado en la escena es capturado por ambas cámaras. A la derecha, la línea formada apilando el píxel capturado a lo largo de toda la pista de cámara.

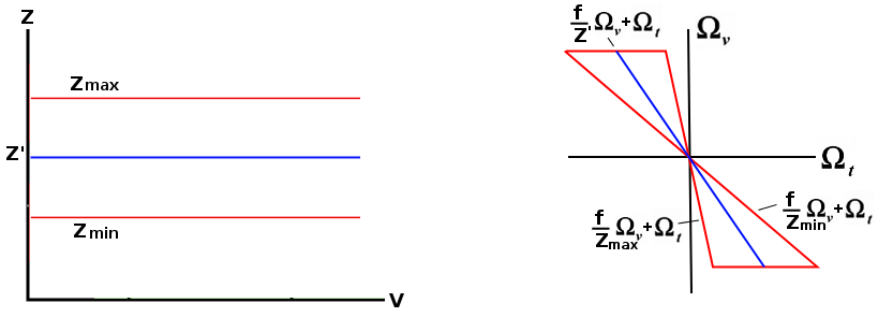


Figura 3.2: A la izquierda, un modelo de profundidad constante. A la derecha, la representación del dominio de frecuencia con los límites para la profundidad mínima y máxima.

de EPIs con su 2D transformada rápida de Fourier (FFT2D) se muestran en Fig. 3.3.

3.2 Análisis de la transformación de dominio de las imágenes EPI

Las Epipolar Plane Images reconstruidas con todas las vistas disponibles representan el escenario horizontal parallax-only perfecto. No hay información perdida. Sin embargo, una de las metas de esta tesis es reconstruir las vistas originales con un número inferior de cámaras. Una manera directa de generar esta propuesta se basa en decimación, lo que directamente descarta filas de forma explícita. Por ejemplo, una decimación de factor 2 descartaría las filas partes obteniendo una imagen final con la mitad de resolución vertical, y una decimación de factor 4 descartaría 3 filas por cada fila que queda, etc.. Aunque este método para manipular imágenes está claro y puede parecer inicialmente correcto, introduce aliasing. Por lo tanto, se requiere un filtro anti-aliasing.

El filtro es del tipo paso-bajo, y discrimina de la imagen la información correspondiente a la alta frecuencia. Producirá algunos cambios visibles en las regiones con colores planos

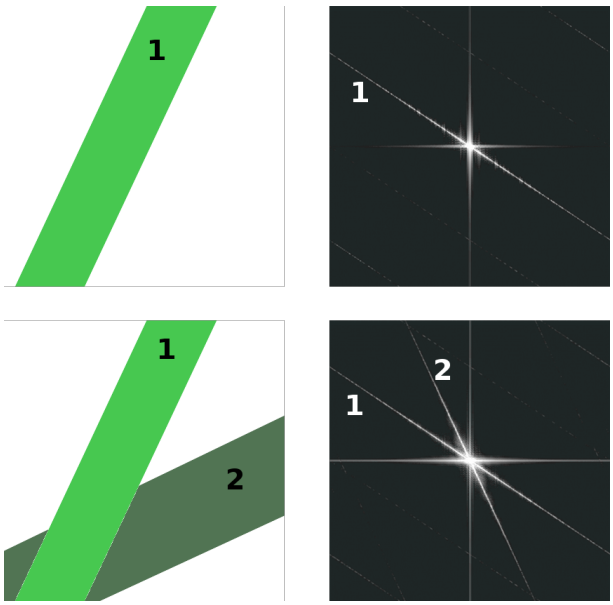


Figura 3.3: La imagen superior muestra un ejemplo de EPI y su transformada de Fourier. La imagen inferior, con un objeto detrás del de la primera imagen, esta más alejado de la escena, y su transformada de Fourier lo muestra en su inclinación.

cercanas a cambios abruptos en el color. Como un efecto colateral, suaviza la aspereza en los bordes. La pérdida de información en el pre-filtrado vale la pena con respecto a los beneficios que proveerá con interpolaciones posteriores, como los resultados probarán cuando se apliquen a imágenes con anti-aliasing.

Dada una imagen (*img*), se aplica el filtro por columnas. Con la intención de describir la esencia de esta técnica, se ha considerado una señal unidimensional *sgn*, representando una columna de la imagen. Fig. 3.4 muestra este proceso en más detalle.

Primeramente, la imagen necesita ser expandida verticalmente por el doble del factor de interpolación utilizado. El primer y último valor de la señal son extendidos hacia el principio y hacia el final correspondientemente. La razón de esta modificación reside en la propiedad circular de la transformada de Fourier. Si esto no se tiene en cuenta, los primero píxeles interpolarían con los últimos píxeles, produciendo un efecto indeseado como un espejo en los extremos.

Una vez la señal *sgn* está preparada, se aplica la Transformada Discreta de Fourier.

$$sgn\_fourier = DFT(sgn)$$

La señal está ahora en el dominio de frecuencia. En este ejemplo se ha considerado un factor de interpolación 2, y por lo tanto, la mitad de la señal tiene que ser filtrada. El filtro remueve la mitad de las muestras correspondientes a la alta frecuencia. Además de este filtrado, la señal tiene que ser normalizada, porque no debe de contener la misma energía. Las fórmulas siguientes representan la teoría detrás de este método, no el código

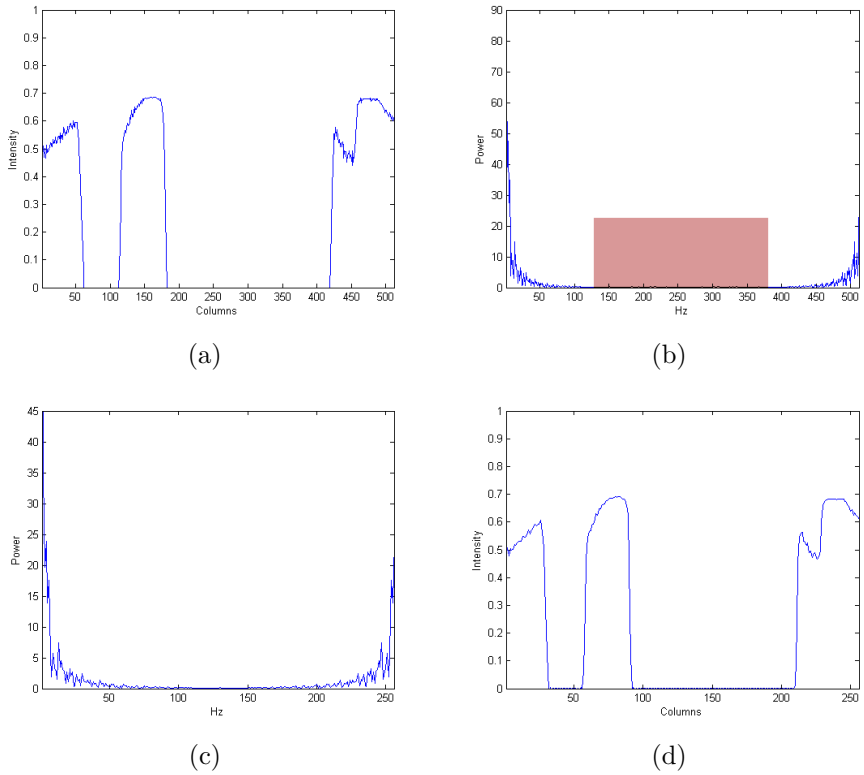


Figura 3.4: a) Señal original con 512 muestras. b) Transformada de Fourier de la señal. La zona resaltada contiene la región de alta frecuencia que tiene que ser filtrada. c) Transformada de Fourier filtrada e interpolada. d) Señal final, la cual ha sido interpolada y suavizada una vez es aplicado el filtro de paso-bajo.

desarrollado completo.

$$normalization = \frac{N.ofcameras}{\frac{N.ofcameras}{2} + 2}$$

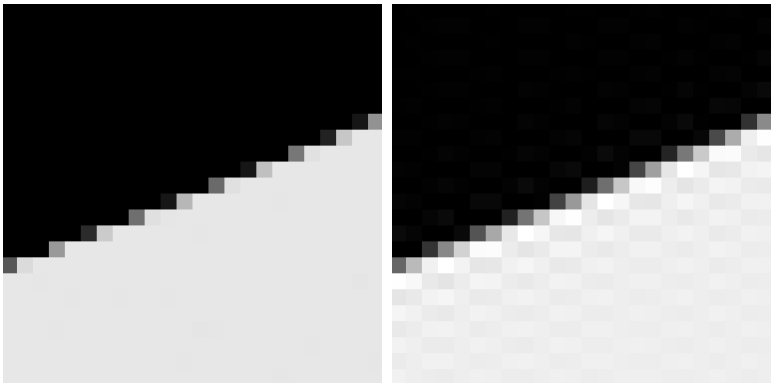
$$sgn\_decimated = \frac{sgn\_fourier(1 : \frac{N.ofcameras}{4} + 2, \frac{3}{4}N.ofcameras + 1 : end)}{normalization}$$

Después de este paso, la señal está filtrada en el dominio de frecuencia. La fase final incluye la transformada inversa de Fourier, y descartar las muestras añadidas inicialmente a los extremos.

$$sgn\_final\_extended = IDFTsng\_decimated$$

$$sgn\_final = sgn\_final\_extended(2 : end - 2)$$

Fig. 3.5 compara dos imágenes epipolares con y sin pre-filtrado. No obstante, se ha aplicado la decimación directa con el propósito de comparar los resultados en la reconstrucción de vistas con ambos métodos.



(a) (b)

Figura 3.5: a) Región detallada de una EPI donde el aliasing es visible. b) La misma EPI con filtro anti-aliasing.

3.3 Filtrado de EPI y optimización del número de vistas

En este momento del proceso, las vistas han sido convertidas en EPIs con cinco factores de interpolación diferentes. El proceso real de optimización tiene lugar ahora, donde las imágenes epipolares pre-filtradas son re-muestreadas a su resolución natural, y se recuperan las filas omitidas inicialmente.

Cuatro de las interpolaciones usadas en esta fase son kernels de interpolación típicamente disponibles en Matlab: triangle (bilinear), cubi (bicubic), lanczos2 y lanczos3. Se ha implementado un nuevo algoritmo: fast DCT-based scaling algorithm basado en el trabajo de [10].

La fórmula siguiente describe el kernel de interpolación desarrollado. Se ha aplicado a señales 1D. Cuando se aplica este método a imágenes (señales 2D), se considera cada columna como una señal individual.

$$signal_k = \sum_{n=0}^{N-1} a_n \{ \quad sincd\left(2N^\circ - 2, 2N, \frac{k+1/2}{factor} - n - 1/2\right) cos\left[\frac{\pi}{2N}\left(\frac{k+1/2}{factor} - n - 1/2\right)\right]$$
$$+sincd\left(2N^\circ - 2, 2N, \frac{k+1/2}{factor} + n + 1/2\right) cos\left[\frac{\pi}{2N}\left(\frac{k+1/2}{factor} + n + 1/2\right)\right]\}$$

donde el término entre llaves representa un kernel de interpolación:

$$N^\circ = min\left(N, \lfloor factor N \rfloor\right)$$

$$sincd\left(M, N, x\right) = \frac{sin\left(\frac{\pi M}{N} x\right)}{N sin\left(\frac{\pi}{N} x\right)}$$

En las fórmulas previas, N representa la longitud de la señal; factor, el factor de interpolación elegido; n es la muestra procesada en cada bucle del sumatorio; y k, la muestra de salida obtenida. Esta propuesta tiene que ser aplicada para cada muestra de salida (k) de la señal, para calcular su valor interpolado. Con la ayuda de Matlab estas operaciones pueden llevarse a cabo más fácilmente, usando operaciones vectoriales y matriciales, especialmente cuando es hecho en dos dimensiones.

El método comentado ha sido desarrollado y testado y se mostrará en el capítulo de resultados. Sin embargo, es fácil descubrir rápidamente algunas limitaciones. El algoritmo acepta cualquier factor mayor que 1, pero diferentes de (1.4, 4.8, 2.2, 2.6, 3, 3.4, 3.8, 4.2, 4.6, 5, 5.4, 5.8, 6.2) y cualquier otro factor impar. Para estos valores, el deenominador en *sincd* devuelve valor infinito. Este método funciona bastante bien con funciones sinusoidales, pero no resulta tan aceptable cuando se trabaja con imágenes con bordes cortantes como el siguiente capítulo mostrará. Esto es un inconveniente dado que las imágenes epipolares contienen señales de este tipo.

Los métodos de interpolación han re-muestreado las imágenes epipolares, pero no pueden ser comparadas directamente a las EPIs originales. Primero las vistas finales tienen que ser reconstruidas, lo que se consigue con el proceso inverso a cómo las EPIs han sido creadas. En vez de seleccionar la misma fila de cada vista para crear la imagen epipolar, se selecciona la misma fila de cada EPI, y se obtiene la nueva vista. La fase de diseño demostrará cómo obtener las vistas reconstruidas a partir de 600 EPIs de 512 píxeles de resolución vertical, y entonces obtener las 512 vistas finales con 600 píxeles de resolución vertical.

La comparación de EPIs originales y reconstruidas arrojarán pruebas de cómo de precisa es la reconstrucción, y consecuentemente, el grado de interpolación que puede ser aplicado en relación con el ruido que aparece. La medición utilizada para analizar esta relación es PSNR (Peak Signal-to-Noise Ratio), conocida como el ratio entre la potencia máxima de una señal de acuerdo con la potencia del ruido presente en la imagen original y comprimida. La compresión, entendida esta vez como el re-muestreo ejecutado en las imágenes epipolares, y más tarde trasladado a las vistas de la escena.

PSNR contiene otro método de medición en su algoritmo, Mean Squared Error (MSE), que mide la media de los errores cuadrados. Una imagen original tiene un Mean Square Error igual a cero, y por lo tanto, infinito PSNR. Cuanto más alto es el PSNR, la calidad de la imagen es mejor.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [Original(i,j) - Interpolated(i,j)]^2$$

con imágenes de tamaño mxn. La fórmula final del PSNR es:

$$PSNR = 10log_{10} \left(\frac{MAX^2}{MSE} \right)$$

donde MAX es el máximo posible valor de píxel de la imagen (255 para las imágenes

usadas). Cuanto más alto es el valor del PSNR, mejor es la calidad de la interpolación, lo cual es explicado con más detalle en el último capítulo.

4. Diseño del sistema multi-vista

Este capítulo examina las técnicas usadas para obtener los elementos necesarios para construir un sistema de holograma generado por ordenador. Las siguientes secciones explicarán las herramientas desarrolladas para obtener apropiadamente las vistas de una escena virtual 3D, el post-proceso ejecutado a ellas y las técnicas de re-muestreo que consiguen la deseada optimización en cuanto al número de vistas.

Figura 4.1: 1) Rendering y recortado de las imágenes para obtener la parte útil de ellas. 2) Creación de EPIs de densidad completa y pre-filtradas para obtener EPIs anti-aliased y re-muestradas. 3) Interpolación vertical de las EPIs para recuperar su resolución original. 4) Reconstrucción de las vistas a partir de las EPIs interpoladas.

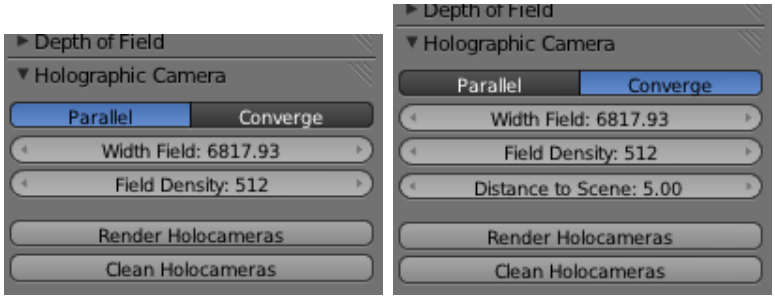
A través de las fases de desarrollo y testado, se han creado más de 32.000 ficheros y se han manipulado casi 3GB de información, lo que requiere una organización de directorios clara y estructurada:

```
/.
functions.blend/py/m (1)
scene_name/epis_density (2)
scene_name/epis_density_interpolation (3)
scene_name/parallel_r_density (4)
scene_name/parallel_rc_density (5)
scene_name/parallel_density_interpolation (6)
```

1) El directorio representa todas las funciones scripts de Blender, Python y Matlab.
2) El directorio contiene las EPIs de densidad completa y pre-filtradas con 5 factores de interpolación.
3) Cada directorio contiene las EPIs interpoladas con 5 factores de interpolación.
4) El directorio contiene los renders originales.
5) El directorio contiene las vistas recortadas a la resolución adecuada.
6) El directorio contiene las vistas reconstruidas para cada factor de interpolación. En el árbol de directorios presentado, densidad es representado por un número (512 en este caso) e interpolación por uno de los 5 kernels de interpolación usado.

4.1 Construyendo el sistema multi-vista para escenas 3D

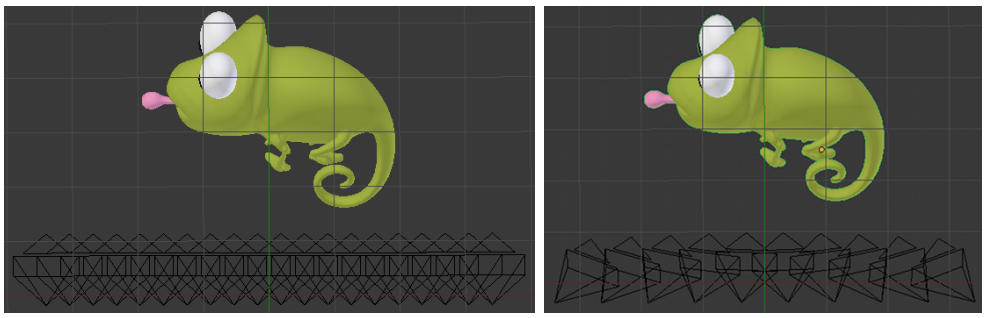
El primer paso antes de analizar las vistas de la escena implica obtener dichas vistas. El software utilizado es Blender como escena virtual 3D y configuración de cámaras. El script que coloca las cámaras en su posición correcta está escrito en Python, e incluye una interfaz gráfica (Fig. 4.2.



(a) (b)

Figura 4.2: a) Interfaz para el modo paralelo. b) Interfaz para el modo convergente.

Inicialmente, el script desarrollado ofrece dos posibles configuraciones, incluyendo cámaras paralelas y convergentes (Fig. 4.3. En capítulos anteriores ya se ha comentado la razón por la que las cámaras convergentes no proveerán las propiedades geométricas apropiadas. Sin embargo, se ha desarrollado este modo con el objetivo de obtener pruebas gráficas de la distorsión producida y estudiar sus consecuencias en el dominio epipolar.



(a) (b)

Figura 4.3: a) Posicionamiento de las cámaras en el modo paralelo. b) Posicionamiento de las cámaras en el modo convergente.

Los atributos que definen la escena están relacionados con los siguientes parámetros:

- **Mode:** Cambia entre el modo paralelo y convergente.
- **Width Field:** Longitud total de la pista de cámara (Unidades Blender * 1000).

- **Field Density:** Número total de cámaras.
- **Distance to scene (Disponible sólo en el modo convergente):** Distancia al centro de la escena donde las cámaras enfocarán.

De acuerdo al modo paralelo, las cámaras se mueven a lo largo de la pista de cámara, capturando vistas paralelas a la escena. La cámara en el centro de la pista tiene el campo de visión necesario para abarcar la escena entera. Sin embargo, cuando la cámara se mueve de un lado a otro, la vista capturada contiene información no necesaria. Fig. 4.4 muestra cómo información vacía es grabada en vez de píxeles pertenecientes a la escena.

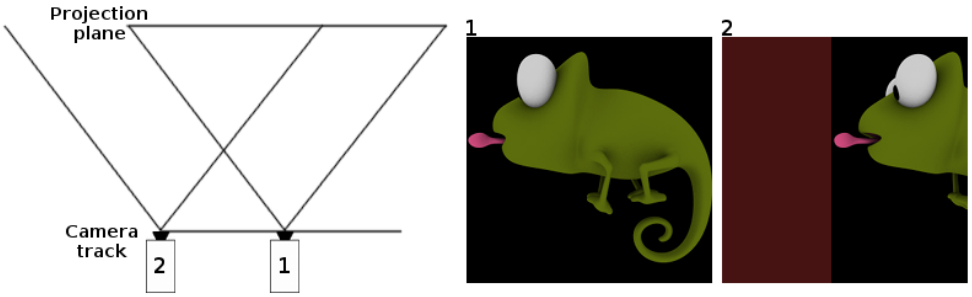


Figura 4.4: La cámara número 2, en el extremo de la pista, captura información no necesaria (en rojo).

Una manera simple de resolver este problema es ensanchar el campo de visión de la cámara, adquiriendo la porción entera del plano de proyección desde cualquier punto de vista. Esta solución requiere más información que renderizar, aunque las regiones adicionales renderizadas suelen contener pocos o ningún objeto, lo que no implica apenas extra tiempo de rendering. Una vez todas las vistas han sido renderizadas, es el momento de post-procesarlas y convertirlas en vistas con la resolución adecuada. Un script escrito en Matlab (desde ahora en adelante, todos los scripts usados para procesar imágenes estarán basados en Matlab) recorta la imagen manteniendo solo la región correspondiente con el plano de proyección. Las imágenes procesadas tendrán la misma resolución después de este recortado. Fig. 4.5 muestra cómo el siguiente script (crop_renders.m) transformará los renders iniciales en imágenes adecuadas para procedimientos posteriores.

```
function crop_renders(object,density,h_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% h_res: Horizontal resolution of views
% It crops renders to normal resolution
...
for i=1:density
    img = imread(strcat(name_in,int2str(density-i+1)),'png');
    cropped = img(:,(i-1)*2+1:(i-1)*2+h_res,:);
imwrite(cropped,strcat(name_out,int2str(density-i+1),'.png'));
end
```

Figura 4.5: La cámara se caracteriza por un campo de visión más ancho. El área innecesaria de la imagen es descartada en la imagen de la derecha.

La escena trabajada en este trabajo contiene 512 vistas, y los scripts de rendering producen imágenes de 1622x600 píxeles. Esas imágenes están recortadas a una resolución final de 600x600 píxeles. La siguiente Fig. (4.6) contiene algunos ejemplos producidos.

Figura 4.6: De izquierda a derecha, muestras de las vistas número 1, 64, 128, 192, 256, 320, 384, 448 y 512.

4.2 Creación y pre-filtrado de EPIs

Todas las vistas obtenidas en la sección anterior componen la representación espacial. Para conseguir una representación del campo de luz de la escena, las vistas tienen que ser transformadas en imágenes epipolares.

Los siguientes pasos incluyen la creación de las ya mencionadas imágenes epipolares y el pre-filtrado de esas imágenes para remover el aliasing. El primer script se encarga de crear EPIs de densidad completa, las cuales tienen el mismo número de columnas que número de vistas renderizadas de la escena (512 en este caso). Un total de 600 diferentes EPIs serán creadas, dado que las vistas tienen 600 píxeles de resolución vertical. Esto es completado por el siguiente script, el cual toma la misma fila de cada vista para crear una única EPI. Por ejemplo, para crear la primera EPI, asigna la primera fila de la primera vista a la primera fila de la EPI, la primera fila de la segunda vista a la segunda fila de la EPI, etc.

```
function epi(object,density,v_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% v_res: Vertical resolution of views -> Number of EPIs
% It creates epipolar images with full density
```

Como resultado de este script en esta particular escena, las primeras y últimas 30-40 filas son completamente negras, dado que ningún objeto aparece al comienzo o final de las vistas. Fig. 4.7 muestra algunas muestras obtenidas por esta función.

Figura 4.7: Desde la izquierda a la derecha, muestras de las EPIs número 1, 100, 200, 300, 400, 500 y 600.

Como ha sido explicado, el filtro anti-aliasing es requerido antes de la decimación. La función presentada más abajo (una versión simplificada) aplica un filtro anti-aliasing. Funciona con 5 factores de interpolación diferentes [2, 4, 8, 16, 32] todos potencia de 2, lo que facilita el procesado de transformadas de Fourier. El re-muestreado produce una salida mostrada en Fig. 4.8.

```
function prefiltering(object,density,v_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% v_res: Vertical resolution of views -> Number of EPIs
% It filters EPIs with anti-aliasing filter
% and down-samples vertically in 5 factors [2, 4, 8, 16, 32]

%Preparation of extended signal
epi = im2double(...);
epi_ext(1+factor:end-factor,:,:) = epi;
epi_ext(1:factor,:,:) = epi(1,:,:);
epi_ext(end-factor+1:end,:,:) = epi(end,:,:);

%Fourier transform, filtering and normalization
epi_f = fft(epi_ext);
normal = density/((density/factor) + 2);
fraction = density/(2*factor);
epi_fdec = [epi_f(1:fraction+2,:,:); epi_f(end-fraction+1:end,:,:)] / normal;

%Inverse Fourier transform and cropping extended samples.
epi_factor = ifft(epi_fdec);
epi_final = epi_factor(2:end-1,:,:);
```

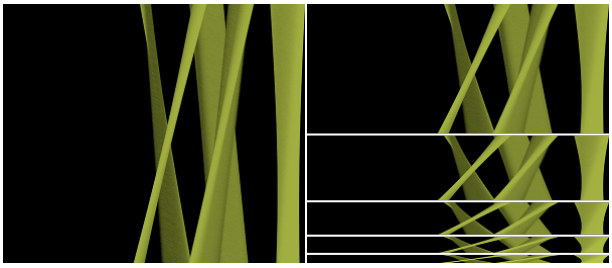


Figura 4.8: La EPI original y las 5 EPIs anti-aliased con sus 5 factores de interpolación diferentes.

4.3 Optimización del número de vistas

En este punto del diseño, los siguientes desarrollos trabajarán en las vistas convertidas en EPIs filtradas, para convertirlas de nuevo a su resolución original. El re-muestreado que optimiza el número de vistas para construir el sistema multi-vista.

Los scripts responsables de esta fase son presentados a continuación:

```
function run_epi_interp(object)
% Object: Name of the scene
% It runs epi_interp() with 5 factors [2, 4, 8, 16, 32] and
% 5 interpolation kernels [triangle, cubic, lanczos2, lanczos3, fastdct]

function epi_interp(object,density,factor,v_res,interpolation)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% Factor: Interpolation magnitude
% v_res: Vertical resolution of views -> Number of EPIs
% interpolation: Interpolation kernel used
% It up-samples vertically EPIs
```

La función epi_interp utiliza la función de Matlab imresize() con los kernels de interpolación bilinear, bicubic, lanczos2 and lanczos3, o la siguiente función fdctEPI, la cual implementa el algoritmo de escalado fast DCT [10].

Este algoritmo ha sido implementado para escalar señales 1D, y más tarde adaptado a dos dimensiones. Todos los scripts codificados que mejoran el código en cuanto a tiempo de computación son presentados en los apéndices. El ejemplo siguiente (versión simplificada) tiene como objetivo demostrar y presentar los fundamentos básicos de la interpolación fast DCT.

```
function img_out = fdctEPI(img_in,factor)
% img_in: input image
```

```
% factor: interpolation magnitud
% It up-samples vertically img_in using fastDCT interpolation

No = rows_in;
n = repmat(0:rows_in-1,rows_out,1);
k = repmat((0:rows_out-1)',1,rows_in);

DCT = sincd(2*No-2, 2*rows_in, (k+1/2)/factor -n -1/2) .* ...
cos( pi/(2*rows_in) * ( (k+1/2)/factor -n -1/2) ) + ...
sincd((2*No-2), 2*rows_in, (k+1/2)/factor +n +1/2) .* ...
cos( pi/(2*rows_in) * ( (k+1/2)/factor +n +1/2) );

% Multiplication by DCT-Matrix to all image component layers
for i=1:components
img_out(:,:,i) = DCT*img_in(:,:,i);
end

% Truncating negative and greater than 1 values
img_out(img_out<0) = 0;
img_out(img_out>1) = 1;

function result = sincd(M,N,x)
% sincd(M,N,x) = sin(piMx/N)/(Nsin(pix/N))
% Digital sinc function used in fastDCT
```

Una de las secciones críticas en el código de fdctEPI() es la población de n y k, dado que representan las matrices que substituyen los bucles iniciales, que operan por filas y columnas. Ejecutar esta operación con dos bucles anidados hace la computación ineficiente. En una versión preliminar, uno de los bucles fue substituido por una operación vectorial, pero popular dos matrices y efectuar operaciones lineales con ellas es más coherente.

Matriz k y n responden al siguiente patrón:

$$k = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ n & n & \cdots & n \end{pmatrix} n = \begin{pmatrix} 0 & 1 & \cdots & n \\ 0 & 1 & \cdots & n \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & n \end{pmatrix}$$

Tres métodos de población de matrices fueron considerados.

```
% Kronecker tensor product
ns = kron([1:size_in]-1,ones(1,size_out)');
ks = kron(ones(1,size_in),[1:size_out]'-1);
```

```
% Replicating matrices
ns = repmat([0:size_in-1],size_out,1);
ks = repmat([0:size_out-1]',1,size_in);

% Manual with ones
temp= [0:size_in-1];
ns= temp(ones(1,size_out),:);
temp= [0:size_out-1]';
ks= temp(:,ones(1,size_in));
```

Los resultados para unas resoluciones de 300, 600 and 900 píxeles y factores 2-10 se muestran en la Fig. 4.9.

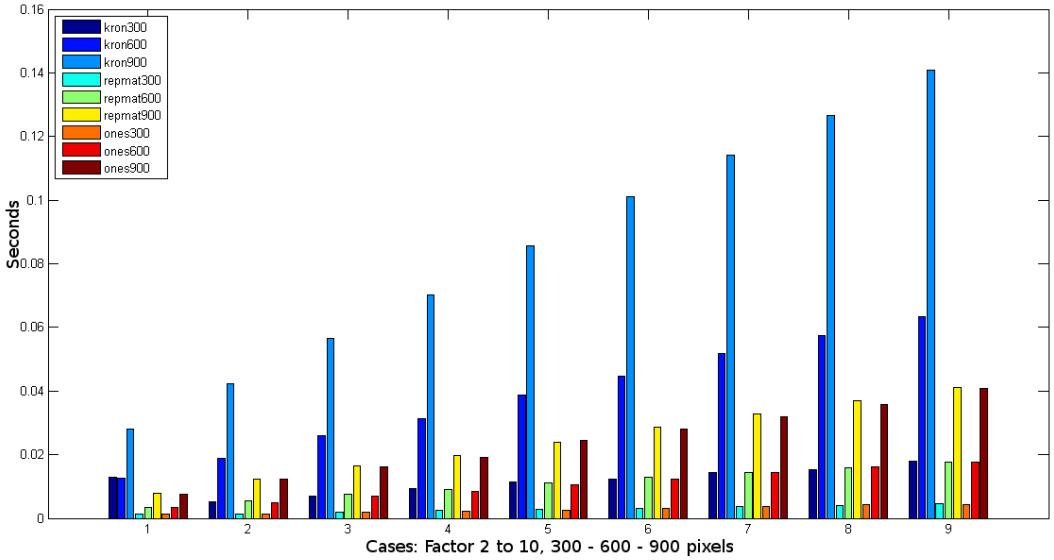


Figura 4.9: El método kron es notablemente peor que los otros dos. No hay grandes diferencias entre el manual y ones, pero al mismo tiempo, manual requiere menos memoria y número de instrucciones.

La reconstrucción de vistas desde las EPIs interpoladas usando los kernels explicados arriba es directa. Una vista es formaba apilando filas que provienen de la misma fila de cada EPI. Las siguientes funciones son las responsables de la fase final.

```
function run_reconstruct_views(object)
% object: Name of the scene
% It runs reconstruct_views() with 5 factors [2, 4, 8, 16, 32] and
% 5 interpolation kernels [triangle, cubic, lanczos2, lanczos3, fastdct]

function reconstruct_views(object,density,factor,v_res,interpolation)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
```

```
% Factor: Factor of interpolation applied vertically
% v_res: Vertical resolution of views -> Number of EPIs
% interpolation: Method/kernel used for re-sampling
% It reconstructs all the views from the re-sampled EPIs
```

La Fig. 4.10 presenta el proceso completo visto desde una perspectiva gráfica.

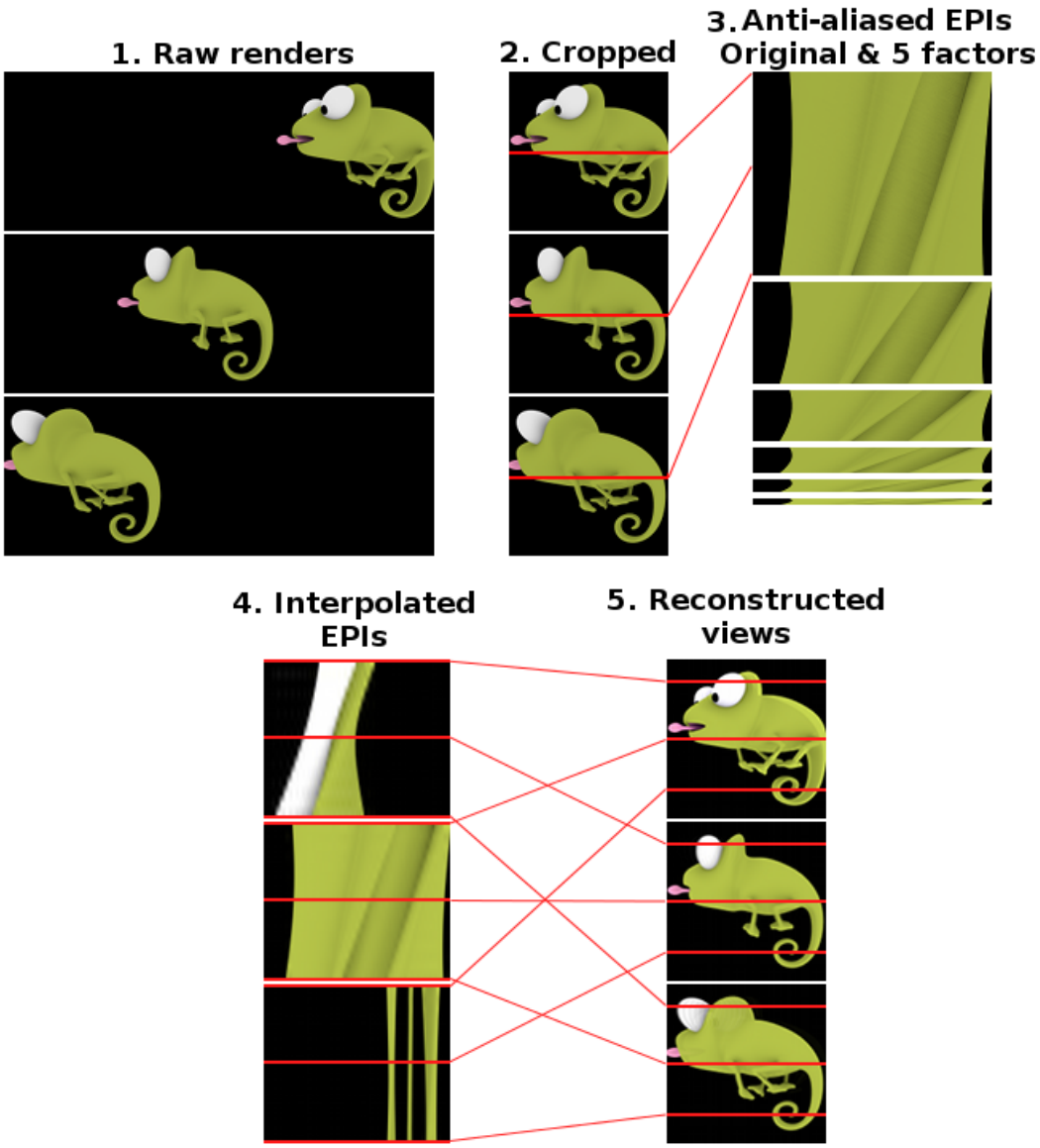


Figura 4.10: La imagen muestra todos los pasos seguidos en este trabajo, y cómo las imágenes son transformadas y manipuladas en cada fase.

5. Resultados

En este capítulo se presentan e interpretan todo los resultados gráficos y numéricos. Los tests principales ejecutados en este trabajo incluyen el algoritmo fast DCT [10] y otros kernels de interpolación, así como la comparación final entre las vistas originales e interpoladas.

5.1 Algoritmo Fast DCT

El algoritmo Fast DCT ha sido usado para re-muestrear verticalmente Epipolar Planar Images. Previo a los tests de imágenes, un set de señales 1D fueron testadas para un mejor entendimiento de los efectos producidos en ciertos casos. Fig. (5.1) muestra alguno de los efectos que el escalado produce en dichas señales.

Consecuencia de este efecto, los bordes de las imágenes epipolares que contienen básicamente lineas y franjas, presentan desenfoque y artefactos. Otro ejemplo, testado para confrontar los resultados de [10] presenta iterativamente el escalado de una imagen de texto. Los resultados arrojan la impresión de que independientemente del número de iteraciones ejecutadas, el texto permanece inteligible, a diferencia de lo que ocurre con otros métodos de interpolación. Fig. 5.2 muestra claramente la diferencia entre dichos métodos.

As it was mentioned above, the border artefacts present will have to be tackled down in posterior steps. The artefacts appearing beyond the borders will take its utility away. Figure 5.3 depicts why the displayed results are not useful for epipolar image interpolation. Como ha sido mencionado previamente, los artefactos presentes en los bordes tendrán que ser abordados en pasos posteriores. Fig. 5.3 explica por qué los resultados mostrados no son útiles para interpolación epipolar.

Otro aspecto considerado en este texto es el tiempo de computación empleado por cada uno de los métodos. Los resultados presentados en Fig. 5.4 muestran que la implementación de Fast DCT realizada para Matlab es más lenta que las funciones built-in en Matlab, dado que Fast DCT comprende dos funciones externas. El hardware usado en la ejecución de estos tests no influencia los resultados, dado que no se están comparando mediciones con información en otro hardware. Además, cada test fue ejecutado 10 veces

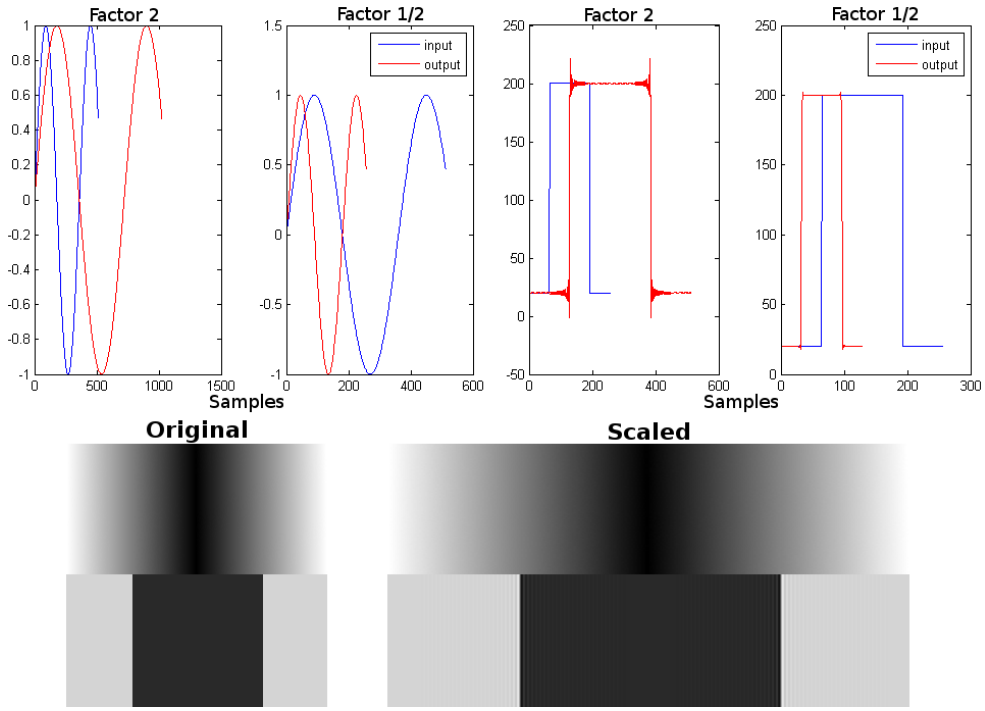


Figura 5.1: El escalado produce en las señales cuadradas un efecto sinusoidal.

(a) (b) (c) (d)

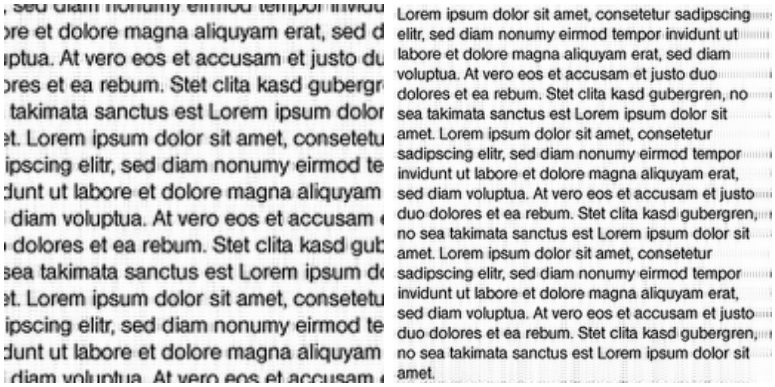
Figura 5.2: a) Imagen de texto original. Escalado iterativo (zoom-in&zoom-out) 75 veces, factor sqrt(2) b) algoritmo bilinear, c) algoritmo bicubic, d) algoritmo Fast DCT.

en las mismas condiciones. Fig. 5.4 muestra la media de todas las ejecuciones.

5.2 Resultados experimentales

El resultado más importante a ser discutido incluye EPIs originales e interpoladas. Esta comparación ha sido llevada a cabo a través de la medición PSNR (Peak Signal-to-Noise Ratio), como se ha explicado en el capítulo de análisis.

El primer resultado experimental con PSNR comprueba cada vista reconstruida para cada método de interpolación. El siguiente script permite obtener dichos resultados:



(a) (b)

Figura 5.3: La imagen de texto escalada 1000 veces con factor sqrt(2) a) Región de la imagen obtenida mostrada como en el estudio [10]. b) Imagen de texto completa, conteniendo los artefactos en los bordes.

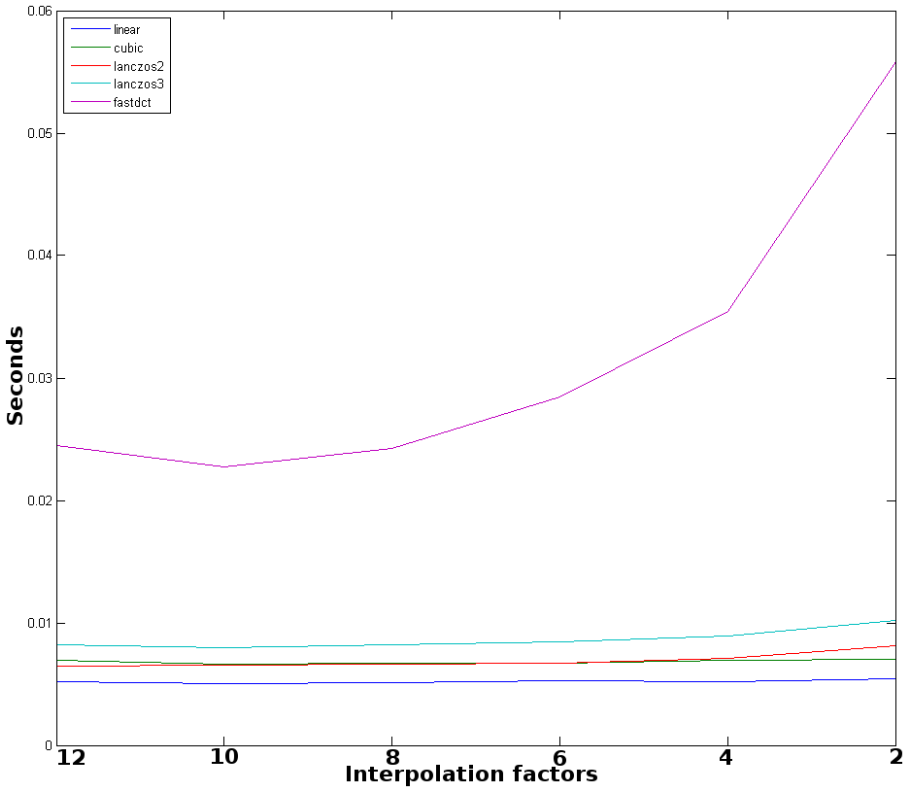


Figura 5.4: La implementación del algoritmo Fast DCT es más lenta que los algoritmos built-in.

```
function test_psnr(object,density,factor)
% Object: Name of the scene
% Density: Number of views
% Factor: Interpolation magnitud
```

```
% It calculates the PSNR for every factor and view
```

Los picos que aparecen en Fig. Figure 5.5 representan las vistas verdaderas no interpoladas que permanecen en la EPI filtrada. Las vistas cercanas a una vista verdadera obtendrán, por lo tanto, mejores resultados PSNR. De acuerdo con esto, las vistas extremas muestras los peores valores, dado que no tienen vecinos a un lado con los que interpolar. Fig. 5.5 pertenece al test con factor 8. Los tests con el resto de factores de interpolación están disponibles en los apéndices.

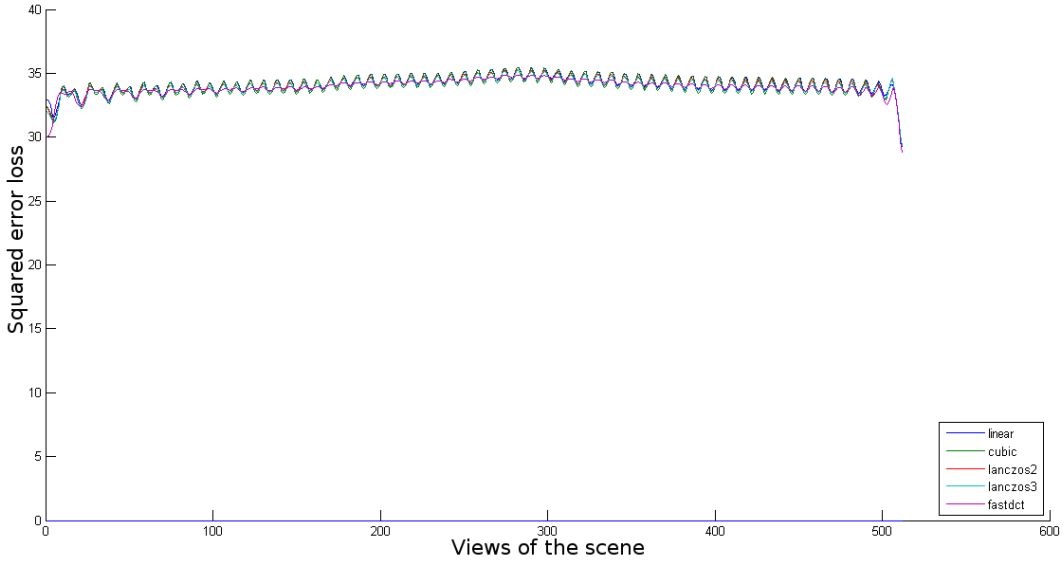


Figura 5.5: Los picos representan vistas originales, y por lo tanto no interpoladas. Las vistas en los extremos tienden a generar peores resultados debido a su isolación.

De igual manera, otro script ha sido escrito para comparar todos los métodos de interpolación en la misma gráfica. El script completo esta disponible en los apéndices.

```
function PSNR = test_psnr_average(object,density)
% Object: Name of the scene
% Density: Number of views
% Factor: Interpolation magnitud
% It calculates the PSNR for every factor and its average between all the
% views
```

La misma gráfica es usada para contrastar los valores PSNR obtenidos en los tests con pre-filtered EPIs y los valores obtenidos con EPIs sólamente decimadas. En el caso de vistas basadas en imágenes epipolares aliased, el factor máximo aplicado ha sido 12, dado que es esperado que con factores mayores los valores PSNR resultarían absolutamente inaceptables.

Vistas basadas en EPIs pre-filtradas son presentadas en la gráfica como puntos, mientras que las no pre-filtradas como líneas. El primer aspecto que se puede notar es que

la diferencia entre diferentes kernels de interpolación no es apreciable en dicha magnitud, por lo tanto, el factor de interpolación será el parámetro decisivo.

A diferencia de como podría haber parecido en el análisis del dominio de Fourier de las EPIs, descartar cierta información relacionada con la alta frecuencia, se ha probado que el filtro anti-aliasing es suficientemente útil. En el próximo capítulo, esta Fig. 5.6 se discutirá más en profundidad. El número mínimo de cámaras necesarias para reconstruir un sistema multi-vista aceptable será determinado en concordancia con esta última gráfica.

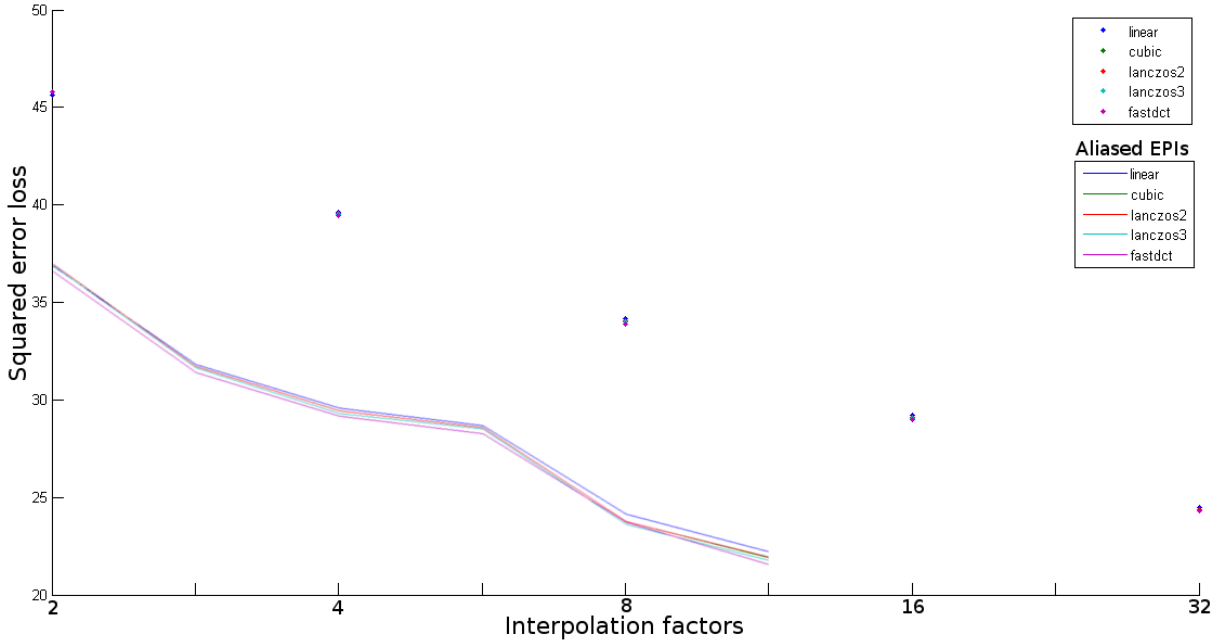


Figura 5.6: Las vistas basadas en EPIs pre-filtradas obtienen muchos mejores resultados para el mismo factor de interpolación que vistas basadas simplemente en EPIs decimadas.

6. Conclusiones

En este trabajo se ha cubierto el objetivo de diseñar herramientas y técnicas que hacen posible un sistema multi-vista basado en una posible escena virtual. Además, las características de la escena han sido analizadas en el dominio espacial y de frecuencia. Comprender estas propiedades nos permite desarrollar nuevos métodos e interpretar el resultado producido durante los tests. Finalmente, un conjunto de tests presentan la eficiencia de cada método y sus limitaciones.

6.1 Conclusiones sobre los resultados

Se ha desarrollado un nuevo filtro basado en el trabajo de [10]. Se ha probado que el filtro funciona mejor para interpolar imágenes de texto iterativamente. Sin embargo, en ciertos casos con bordes, el caso de imágenes epipolares, los artefactos producidos alrededor de los bordes afectan el rendimiento total en el momento de reconstruir las vistas.

El algoritmo Fast DCT ha sido implementado en Matlab, y por lo tanto, es más lento que las funciones built-in.

La optimización de cámaras se ha realizado mediante interpolación vertical de las imágenes. Estas Epipolar Planar Images representan cómo el campo de luz de cada píxel cambia a lo largo de diferentes puntos de vista. Para obtener mejores resultados después del re-muestreo, se ha aplicado un filtro anti-aliasing. Los resultados mostrados en el capítulo anterior prueba que filtrar EPIs beneficia el resultado general.

Los tests ejecutados para determinar la calidad de las vistas reconstruidas están basados en la medición Peak Signal-to-Noise Ratio Las imágenes comparadas están representadas en el espacio de color RGB. Otra opción es convertir dichas imágenes primero en el espacio de color YCbCr y entonces comparar el canal Y (luminancia). La razón para tomar esta decisión reside en la capacidad de la vista humana para percibir mayores perturbaciones y ruido en el brillo que en el color.

Finalmente, con los tests experimentales PSNR, es posible hacer una estimación del número óptimo de cámaras. Fig. 6.1 ayuda a entender esta estimación. No hay un valor exacto del número apropiado de cámaras a elegir, sino un intervalo donde los resultados

serán aceptables con referencia a su aplicación. PSNR no dicta una decisión final. Algunos resultados visuales, percibidos por el sistema visual humano como mejores, pueden conducir a peores valores PSNR. Las interpolaciones son también altamente dependientes de la escena escogida. En este trabajo, el camaleón virtual contiene diferentes formas y superficies, y está situado en una profundidad media.

No obstante, la posibilidad de probar este sistema multi-vista desarrollado junto con la escena virtual interpolada y optimizada en una placa de holograma no ha sido posible. Por lo tanto, la apariencia visual no esta considerada, y el juicio a tener en cuenta es la medición PSNR. Un valor aceptable para imágenes con pérdidas se sitúa entre 30dB y 50dB. Se ha solapado una franja en la figura 6.1. No se ha encontrado una gran diferencia entre los diferentes métodos de interpolación. Como consecuencia directa, la opción recomendada debe de ser la más rápida y simple. El kernel de interpolación triangle, también conocido como bilinear, es el más rápido y provee el mejor valor PSNR, sin embargo, no muy distante del resto.

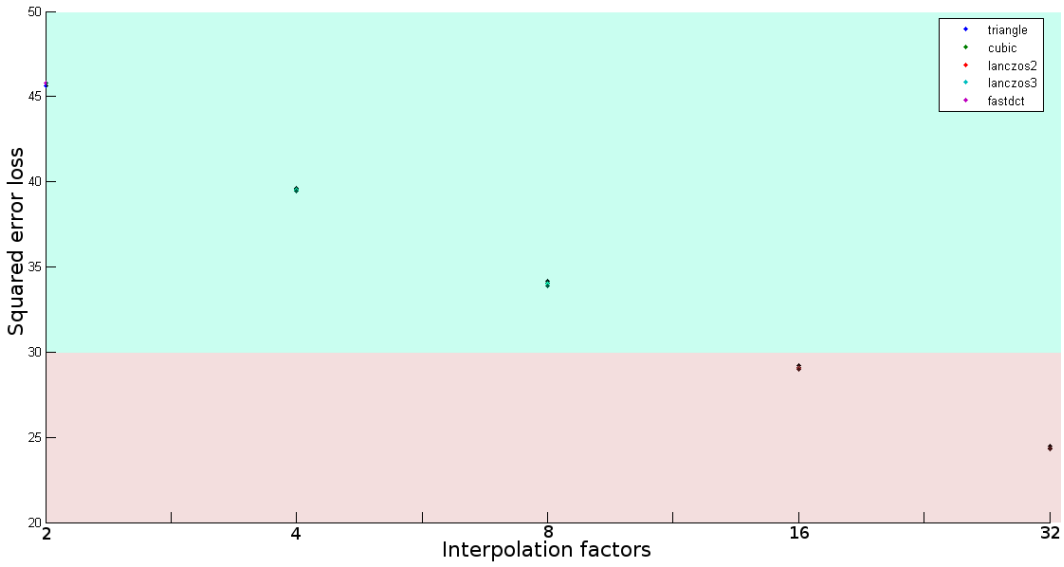


Figura 6.1: La franja azul contiene valores muy buenos para imágenes con pérdidas. La franja roja contiene imágenes con pérdidas para transmisiones wireless, por debajo de los requisitos de calidad para este objetivo.

Consecuentemente, los factores de interpolación que proveen sistemas multi-vista de buena calidad pertenecen al intervalo [2-16]. En otras palabras, usar desde la mitad hasta una dieciseisava parte del número de vistas iniciales. 512 vistas han sido usadas en este trabajo. 32 vistas procesadas con algoritmos anti-aliasing y de re-muestreado permiten reconstruir las 512 vistas iniciales con un valor PSNR de 29dB. Usar una interpolación de factor 32 generaría un valor relativamente bueno (24dB); sin embargo, se necesita establecer un criterio con algunos límites.

6.2 Trabajo futuro

Este proyecto estudia un intento inicial de establecer una metodología para un rendering multi-vista. Usa cámaras virtuales debido a la libertad de manipulación de sus parámetros. También toma ventaja del entorno 3D para renderizar objetos y escenas que pueden ser adaptados de mejor manera a los requisitos establecidos, a diferencia de los objetos en el mundo real.

El siguiente paso a seguir implicaría el uso de cámaras físicas para adquirir las vistas, nuevos métodos de interpolación incluyendo técnicas mixtas. Varias propuestas avanzadas con respecto la configuración de cámaras, siendo estas re-centradas más complejas que lleven a una configuración full-parallax. También, parámetros intrínsecos del las lentes en las cámaras reales jugarán un papel importante en los temas relacionados con la profundidad.

Llevar este estudio a posible aplicaciones es el siguiente paso a dar. Por ejemplo, juegos gráficos por ordenador con interacción en tiempo real requieren más y más simplificados modelos de polígonos, dado que toman la ventaja de mapping de texturas avanzados. Los modelos, más simples, pueden ser manipulados mejor en el dominio epipolar, y la calidad del producto final será más dependiente a nivel de textura.

El trabajo de este proyecto simboliza un número independiente de fases para obtener un resultado final. Sin embargo, un estudio futuro podría desarrollar todos estos pasos en una única unidad de producción. Las especificaciones de dicho software de producción variaría considerablemente, dependiente de las diferentes plataformas, herramientas y software de desarrollo usado durante el proceso. Si una herramienta de producción tiene que ser desarrollada, algunos aspectos tendrían que ser tenidos en cuenta. Por ejemplo, todos los algoritmos deberían de ser escritos en lenguajes compilados, donde una verdadera optimización podría tener lugar. Mex o C podrían ser un buen punto de partida como alternativa a Matlab. Algunas partes del código que requieren una inmensa transferencia de entrada/salida podría ser paralelizados. El aspecto más importante en la optimización de código reside en la Ley de Amdahl: la optimización de un bloque de código optimizará la aplicación general en la misma proporción que ese código representa el proceso general. Algunas veces un gran esfuerzo es puesto en optimizar algo que, como ha ocurrido en el estudio de población de matrices, no refleja un beneficio digno en comparación con el tiempo invertido.

6.3 Valoración personal

Esta tesis ha significado un proyecto completamente especial en el período que ha tenido lugar, por varias razones. Primero, afronté este proyecto en un ambiente internacional, como parte de mi estancia Erasmus en Finlandia en la Tampere university of Technology. En segundo lugar, ser investigador en el departamento de Signal Processing me ha

mostrado una cara de la universidad en la que no había estado involucrado antes. También me ha dado la posibilidad de acudir a la 3D Media Training School en el verano de 2012 en Tampere. Finalmente, aunque mis conocimientos previos son enteramente basados en Computer Science, el reto de adaptar mi conocimiento y comprender algoritmos y propiedades relacionados con el procesado de señal (la mayoría de ellos no estudiados en mi programa con anterioridad) me ha impulsado a afrontar los problemas desde una perspectiva diferente. Además, hacer uso de mi experiencia en software y computación ha sido de ayuda a la hora de desarrollar ciertos algoritmos. Mi visión de cómo las imágenes son procesadas y cómo esa tecnología es usada ha cambiado después de completar esta tesis.

Bibliography

[1] E. N. Leith. *White light holograms*. Scientific American, (1976).

[2] Michal W. Halle. *The Generalized Holographic Stereogram*. Massachusetts Institute of Technology, (1991).

[3] Michal W. Halle, Adam B. Kropp. *Fast computer graphics rendering for full parallax spatial displays*. Brigham and Women's Hospital, Massachusetts Institute of Technology, (1997).

[4] Ravikanth Pappu *et al. A Generalized Pipeline for Preview and Rendering of Synthetic Holograms*. Massachusetts Institute of Technology, Interval Research Corporation, (1997).

[5] Michal W. Halle. *Multiple Viewpoint Rendering*. Brigham and Women's Hospital, (1998).

[6] Jin-Xiang Chai *et al. Plenoptic Sampling*. Microsoft Research China, (2000).

[7] Wendy Plesniak *et al. Reconfigurable Image Projection Holograms*. Brigham and Women's Hospital, MIT Media Laboratory, ThingMagic, Inc., (2006).

[8] Smithwick, Quinn Y. J. *et al. Real-time Shader Rendering of Holographic Stereograms*. Society of Photo-Optical Instrumentation Engineers, (2009).

[9] Melania Paturzo *et al. Holographic Display of synthetic 3D dynamic scene*. 3D Research Center and Springer, (2010).

[10] Leonid Yaroslavsky, Leonid Bilevich. *Fast DCT-based Algorithm for Signal and Accurate Scaling* . Tel Aviv University, (2012).

7. Apéndices

Las secciones siguiente contienen todo el material que no podía ser incluido en el texto principal debido a su extensión o relevancia.

7.1 Original MSc Thesis at Tampere University of Technology

El primer anexo incluye el proyecto realizado originalmente en inglés, en la Tampere University of Technology, Tampere, Finlandia. La numeración de dichas páginas no se ha cambiado para mantener una coherencia con la tabla de contenidos e índice de figuras con las que el documento comienza.

Table of Contents

1 Introduction **46**

1.1 Context and motivation 46

1.2 Scope and objective of the project 50

1.3 Structure 51

2 Background **52**

2.1 Computer generated hologram method 52

2.2 EPI representation 53

2.3 Requirements of the approach 55

3 Analysis **58**

3.1 Relations between scene characteristics and Fourier domain representation 58

3.2 Transform-domain analysis of EPI images 60

3.3 Filtering of EPI and optimizing the number of views 61

4 Design of the multi-view system **64**

4.1 Building a multi-view system from 3D scenes 65

4.2 Creation and pre-filtering of EPIs 67

4.3 Optimizing number of views 68

5 Results **73**

5.1 Fast DCT algorithm 73

5.2 Experimental results 75

6 Conclusions **78**

6.1 Conclusions about the results 78

6.2 Future Work 80

6.3 Personal Assessment 80

List of Figures

1.1 Hologram in Star Wars 47
1.2 Horizontal Parallax Only vs. Full-parallax stereograms 48
1.3 Cross-section of a camera with projection plane 48
1.4 Holographic table layout 49
1.5 Collimating lens 49
1.6 Typical holographic recording scheme 50
1.7 Example of holography reconstruction 50

2.1 Representation of the hogel (hologram element) 53
2.2 Camera arrangement for full or horizontal only parallax 54
2.3 EPI construction process 55
2.4 Example of occlusion in EPI 55
2.5 Four possible configurations for camera arrangement 56
2.6 EPIs obtained with different camera configurations 56

3.1 Depth function represented in spatial and epipolar domain 58
3.2 Spectral support of light field bounded in frequency domain. 59
3.3 EPI representation and 2D Fourier transform 59
3.4 Anti-aliasing steps for a one-dimensional signal. 61
3.5 Comparison between anti-aliasing filtered and non filtered EPIs. 62

4.1 Scheme of developing process 64

4.2 Interfaces for parallel and convergent scripts in Blender 65
4.3 3D example of parallel and convergent cameras in Blender 65
4.4 Cropping problem of views with parallel cameras 66
4.5 Solution for cropping renders. 67
4.6 Samples of views. 67
4.7 Samples of EPIs. 68
4.8 Samples of EPIs filtered with anti-aliasing. 69
4.9 Populating matrices with kron, repmat and ones. 71
4.10 Graphical perspective of the whole process. 72

5.1 Sample signals scaled with Fast DCT algorithm 73
5.2 Comparison iterative scaling between bilinear, bicubic and fast DCT. . . . 74
5.3 Border effects in the DCT method 74
5.4 Time comparison between fast DCT implementation and rest of interpolation kernels. 75
5.5 PSNR of reconstructed views for every interpolation method. 76
5.6 Average of PSNR for every factor against aliased EPIs-based views. 77

6.1 Average of PSNR for every factor and range of acceptable PSNR values. . . 79

1. Introduction

In this work we review the holographic stereogram printing approaches studied thus far and its implementation through a multi-view image rendering system. In this chapter, the fundamental concepts of holographic printing are presented, as well as the motivation to explore this topic, and the structure of the rest of the document.

1.1 Context and motivation

Since the first stereoscopic displays, researches have tried to improve the virtual reality experience bringing more realistic and immersive 3D environments to the customers. The challenge rests here in recording a scene from the real world and displaying the recorded data onto synthetic material. Sometimes, precisely the case we work with in this thesis, the source material does not belong to the real world; it is computer generated, hence, easier to manipulate.

First holographic displays, developed in 1964 by Leith [1], were illuminated by laser. Since then, media and film productions showed holograms as a futuristic way of recording and displaying the real world in real time. The best example of all, first Star Wars movie in 1977 (one year later than Leith's development) already introduced holocameras and holoprojectors as figure 1.1 shows.

A hologram contains the interference pattern produced by the reference and object wavelength, which have different phase. Unlike typical stereoscopic photographs a hologram is both the recording medium and its display. Holograms were disadvantaged by some recording requirements whereas a hologram must be recorded with a monochromatic and coherent light. Furthermore, objects being subject of recording had to stay stable within a fraction of light wavelength [2].

Unlike recent synthetic display holograms [9] traditional holograms required to store a massive amount of data in them, making the computation a heavy process. Holographic stereography supposes that only a limited number of viewpoints are recorded, regarding human perception desired rather than the storage capacity of the hologram. This is a notable difference between the former true hologram. Another great difference means a separated recording process. As any traditional stereogram, images can be captured with

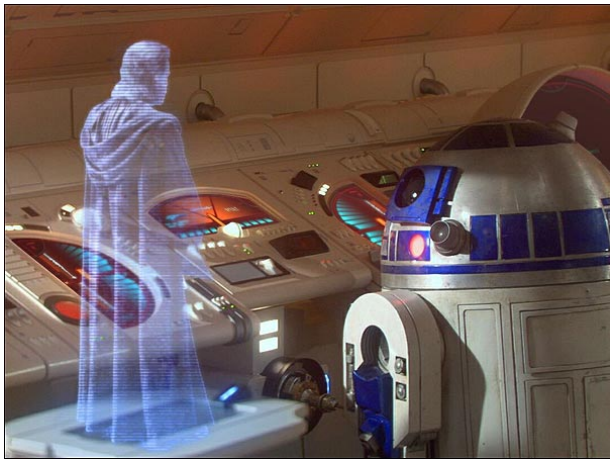


Figure 1.1: Star Wars frame where a hologram is projected from a board below.

an ordinary photographic camera and processed by computer graphic techniques later on. Moreover, we can substitute real life object by computer graphic models, virtually costless to configure.

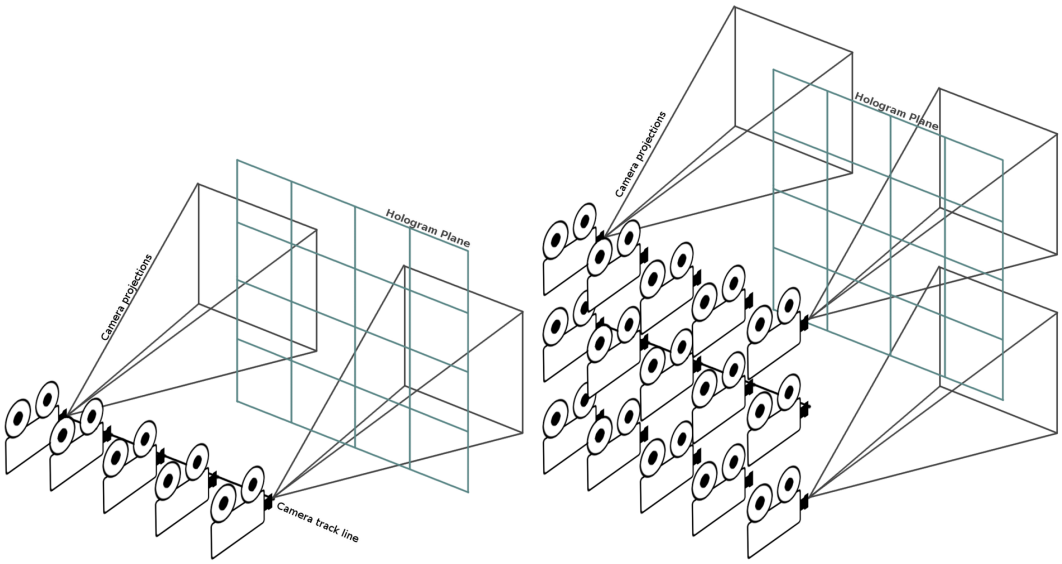
A stereogram is created in three stages [2]:

1. Photographic capture, either from real objects or synthetic 3D models.
2. Holographic recording.
3. Final viewing.

Before going deeper into the holographic recording process, some terms have to be explained to understand distinct approaches relating ways of acquiring views from an object such as Full-parallax and Horizontal Parallax Only (HPO) stereogram. While former attempt to simulate the real world, the latter offers a decent three-dimensional view with much less camera viewpoints. Figure 1.2 illustrates this difference.

All cameras are always pointing ahead, and their axis are parallel to the hologram plane and camera track line. Camera projections allude to pyramids representing their field of view. Given pyramid is intersected by the hologram plane resulting in projection plane of every camera. A more detailed view of the cross-section of the viewing pyramid is shown in Figure 1.3. We use a parallel camera-scene setting to explain how a multi-view system works. Nevertheless, further settings will be discussed later in next chapters.

A holographic plate is composed by vertical slit holograms along the plate. Each slit is exposed to an image projected onto a rear-projection screen, representing a different viewpoint. After the hologram is recorded, every viewpoint can be seen through each slit conforming an aperture. A viewer looking at the stereogram will see through two different apertures, one for each eye. The brain interprets the differences between the two views as three-dimensional information. The appearance of a three-dimensional scene works



(a) (b)

Figure 1.2: a) Scheme of HPO stereogram. Cameras are placed along a camera track parallel to the same hologram plane. b) Scheme of full-parallax stereogram. Cameras are placed along a plane parallel to hologram plane.

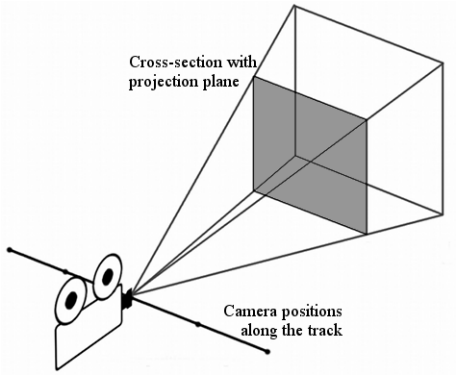


Figure 1.3: In this configuration, the camera always points ahead. Note the cross-section between its pyramid and the projection plane as it is moving along the stereogram track.

according to a binocular depth cue. Both viewer's eyes follow two different slits of the hologram. Images that appeared in those slits fall precisely in front of the eyes, producing the stereoscopic effect of being at infinite. Nonetheless, the observer focus his sight to projection plane, placing the image on there. If the viewer moves laterally, a new pair of images are shown, simulating the 3D scene. As slits are recorded horizontally, following HPO scheme, the viewer sees always the same vertical perspective regardless the vertical position where he observes from. As a matter of fact, the image will move as if it is on the vertical projection plane.

Concerning the holographic recording presented in [2], two beams of coherent light are used. Laser beam is split in two, originating a reference beam and an object beam.

This object beam is enlarged as it passes through the camera and projection lens. It illuminates the hologram by projecting onto the projection screen an image recorded on cine film in the camera. The projection frames represents a 2D object located on the boundless projection plane, and at the same time the projected image represents a visible subarea of the projection frame for any viewpoint. The projection screen and the holographic plate along with the slit mechanism faced each other, and it can be seen in the Figure 1.4 more precisely. The holographic plate is exposed both to the image projected and the reference beam. However, only one stripe of the hologram is open for exposure, considering a opaque piece perforated with slit-shaped holes covering the hologram plate.

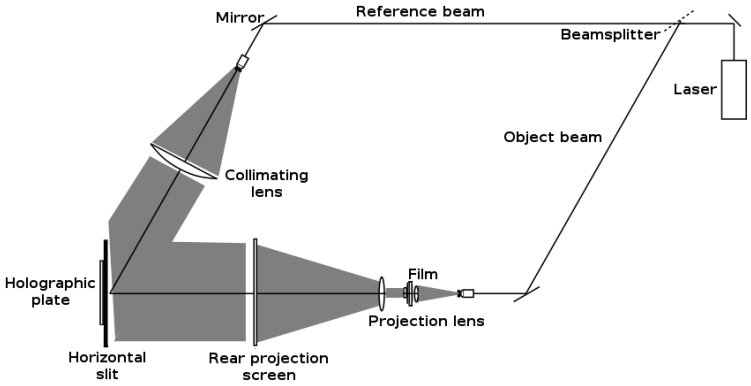


Figure 1.4: Holographic table layout from above (Source [2]).

Each slit has to be illuminated with a reference beam of the same direction, regardless the lateral position of the slit. This is achieved by gathering the beam with collimating lens or Frensel Lens, the same principle used in lighthouses. Figure 1.5 gives a more detailed idea of how a collimator works.

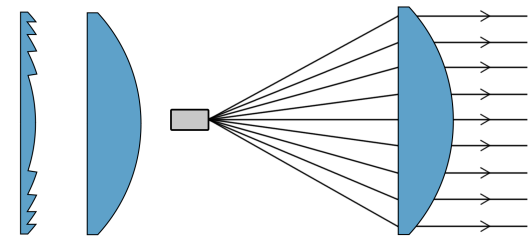
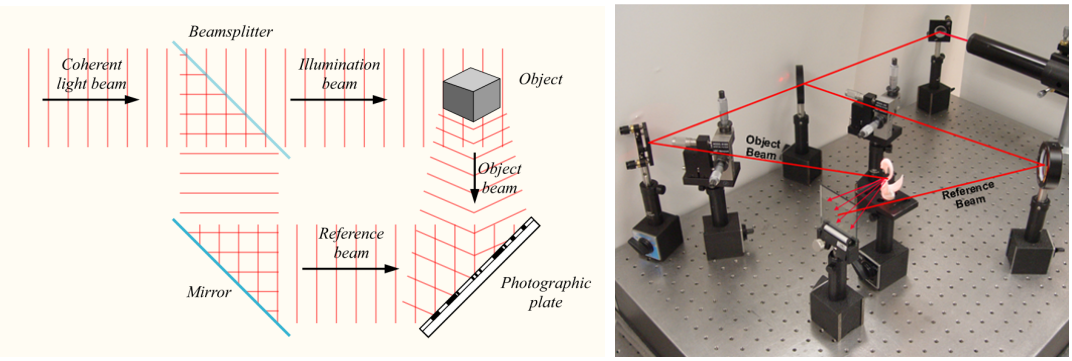


Figure 1.5: a) Fresnel lens used in lighthouses. b) Equivalent plano-convex lens. c) Scheme of collimator that aligns the beams causing a parallel output.

Another typical configuration used for holographic recording consists of casting laser beams at a physical object and recording the interference of wavelengths that the object releases together with the reference beam. The detailed scheme is presented in Figure 1.6. Instead of collimating beams, lenses and mirrors are used to redirect them towards the holographic plate. The recording is done in darkness to avoid any type of interference coming from day light.



(a) (b)

Figure 1.6: a) Scheme of holographic recording with object exposed to laser beams. *Bob Mellish.* b) Photography of the same scenario in a laboratory. *College of Optics & Photonics, University of Central Florida.*

When the holographic plate is illuminated by an equivalent reference beam, a reconstruction of the object can be seen. If the position of the plate or the viewer changes the appearance of the object changes as well as if it was the real object. An example of holographic reconstruction is given in Figure 1.7.

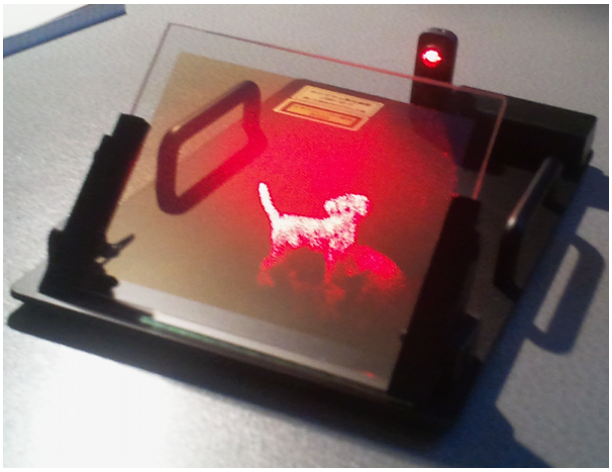


Figure 1.7: The dog is reconstructed with reference laser beam applied to the hologram. *Hologram presented by Dr.Kenji Yamamoto at 3D Media Training School, Tampere, 2012.*

1.2 Scope and objective of the project

The objective of this project is to understand the creation process of a hologram in order to develop novel techniques. Necessary software is developed in this work with the purpose of providing data needed for following steps of the process.

The milestones to be achieved in this thesis include:

- Design and development of a script that provides a set of rendered images corresponding to a horizontal-parallax only system from a 3D scene.
- Creation of epipolar images (EPI) from rendered images, allowing to represent all views into a single image.
- Analysis of EPI images in frequency domain. Relation between scene characteristics and representation in Fourier domain.
- Design and development of a filter to process EPI images, optimizing minimum number of cameras.

1.3 Structure

This thesis is organized with the following structure:

Chapter 2 describes the state of the art related with the topic.

Chapter 3 presents a theoretical analysis which is combined with the experiments.

Chapter 4 explains the tools developed to accomplish the experiments and justifies their purpose.

Chapter 5 shows the results obtained and their graphical representation.

Chapter 6 completes the research providing both the conclusions and the future work that could pursue the present study.

Chapter 7 contains the bibliography references used for the development of this research.

Next chapters contains the appendices including the rest of development accomplished during this work.

2. Background

This chapter reviews the state of art in the area as well as all the necessary knowledge to understand the topic. It overviews the holographic stereogram printing approach.

2.1 Computer generated hologram method

As explained in the previous chapter, a hologram contains light field information. A holographic stereogram is a discretised version composed of several hologram elements called hogels. Every hogel contains 3D information from various perspectives, which was previously recorded depending on the the angle of refraction (i.e: viewer's point of view).

A fully computed hologram offers an infinite number of perspectives. On the other hand, a holographic stereogram offers only a finite number of perspectives instead, using wavefront reconstruction. Holographic stereograms form an approximation of the wavefront, for decreased computation time. This approximation is done with a discrete number of perspectives. The approach developed in this thesis will pursuit the optimization of number the views needed [6], obtaining a similar holographic perception.

One method of diffraction-specific encoding [8] refers to the holographic stereogram as a sum of overlapping amplitude modulated chirp gratings. A chirp signal increases its frequency over time. This amplitude modulated chirped grating on the hologram plane produces a set of view-directional emitters on an emitter plane. Each chirp focuses light to create a point emitter, while the angle-dependent brightnesses of the views are encoded in the amplitude modulation.

According to Quinn [8], the parallel vector computation of a hologram is divided into three steps:

1. Pre-computing the chirp vectors into a chirp texture.
2. Multi-view rendering of the scene directly into the modulation vectors stored in the modulation texture using a double frustum camera.
3. Assembling the hologram by gathering chirp and modulation vectors via texture fetches and then performing a dot product.

This thesis will focus on a portion that is developed during the second, with some variations, e.g., horizontal parallax-only cameras instead of using double frustum. The exact differences will be discussed in Section 2.3 at the end of this chapter, where the requirements of the developed approach are presented.

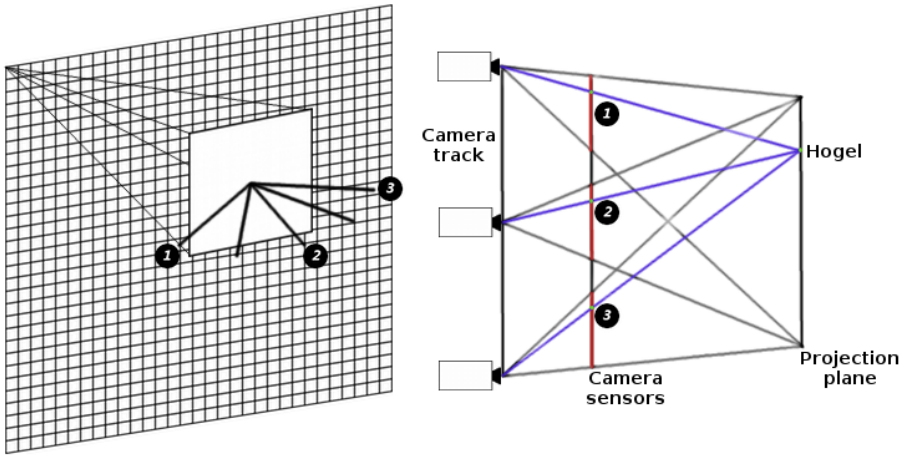


Figure 2.1: On the left, the hologram and an magnified picture of a single hogel. On the right, the pixel stored in the sensor, representing the same pixel of the scene, is recorded in the hogel. The camera sensors (in red color), are placed in the actual camera.

A more general approach to understand how the multi-view rendering system works is based on the hogels introduced previously. Figure 2.1 will picture intelligibly the following explanation. Every hogel contains the light wavefront corresponding to one pixel in the scene, coming from every point of view. This information can be reconstructed applying the proper light to the hogel, then emitting the data recorded depending on the point of view. Several cameras record from different positions along the camera track line. Each sensor is composed of many pixels. A single hogel contains the information associate to the same pixel of every view. It means that it holds data from as many different directions as number of views used in the scene.

2.2 EPI representation

The correlation between images from different positions of the same static scene connotes the perspective coherence. The main distinction between perspective and temporal coherence is that the former is more restricted than the latter, related to changes in images along time. Scene characteristics, i.e: shading and geometric changes, can be analysed in the epipolar domain.

An Epipolar Plane Image (EPI) is a 2D representation of the light field. It constitutes the transformation of a set of multi-view images into a multi-perspective light field cut. In order to obtain this representation, first cameras should be arranged as shown in Figure

2.2 exhibits. Halle [5] describes these two different arrangements as regular shearing (RS) cameras: planar regular shearing (PSR) and linear regular shearing (LSR).

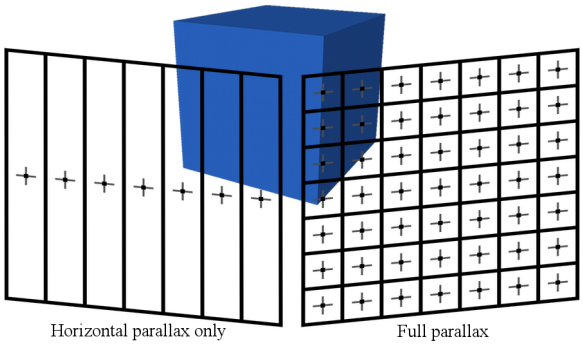


Figure 2.2: On the left, the LSR disposition provides only horizontal parallax while the right dispotition, PSR, captures horizontal and vertical information.

The LSR disposition, which consists of a single array of cameras represents a HPO setting, providing all the necessary information with only one row of cameras, but removing vertical parallax. Viewer will receive the same image as he sees from a different vertical position. On the other hand, the PSR disposition supplies full parallax, but, it requires many more cameras.

Apprehending how EPIs are formed helps to understand the holographic printing system, since EPIs conform an essential element within the process. Imagine that all images rendered from the multi-view system are stacked as deck of cards. The image corresponding to the rightmost view is placed on the front, and the leftmost view on the back. Then, the created spatio-perspective volume is sliced horizontally. The multi-perspective cut obtained, i.e: the EPI, holds the same equivalent row of every view and it is lying down, pointing at the top. In other words, the bottom row of the EPI belongs to the rightmost view and the top row belongs to the leftmost view. Figure 2.3 shows graphically the technique used to obtain the EPI.

Although Figure 2.3 displays a single EPI, all of them should be calculated in order to build the entire spatio-perspective volume. There are as many EPIs as height resolution of the camera plane.

The convenience of using EPIS is strongly notable with regard to possibilities of using some of the features in further developments. For instance, their linear structure allows to effectively apply interpolation algorithms. This is utterly important as one of the goals of this project (explained in more detail in the next chapter) responds to optimizing the number of views. Additionally, EPIs gather extra features, such as side detection, shading and occlusion of the objects. The last one can be observed in the Figure 2.4 where a different row has been chosen.

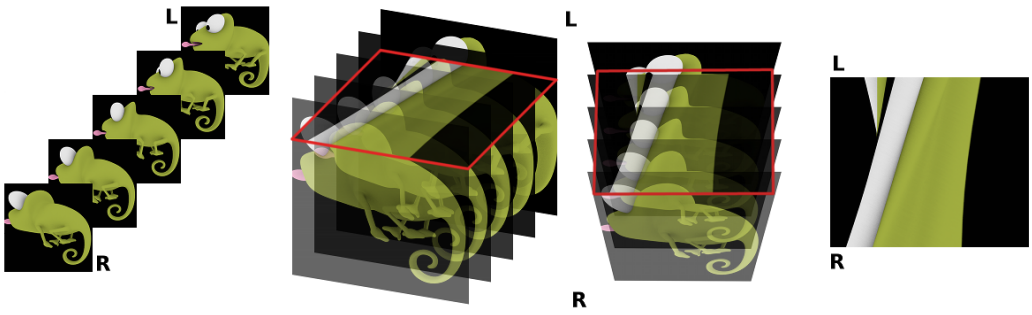


Figure 2.3: On the left, five views representing from the leftmost to the rightmost views, with some intermediate ones in between. In the middle, both 3D perspectives show the views stacked. The red frame designates the row chosen for this EPI, which reposes horizontally. On the right, the resultant EPI contains the leftmost views on the top and the rightmost views on the bottom. Notice how the right eye of the chameleon disappear from the EPI as the camera goes to the right.

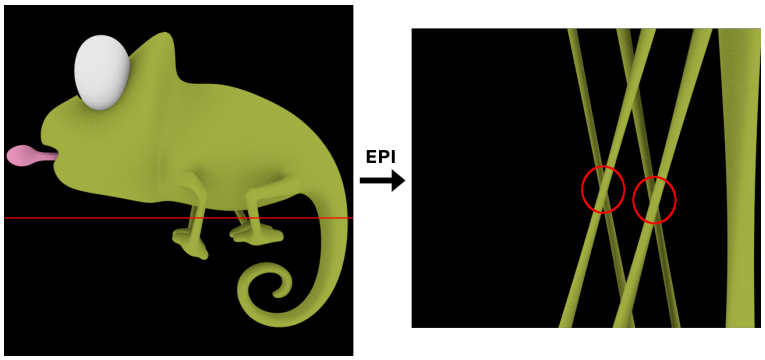


Figure 2.4: On the left, our 3D chameleon and the row chosen marked in red. On the right, the resultant EPI where occlusions produced by both pair of legs are enclosed in a red circle.

2.3 Requirements of the approach

After presenting the necessary background, this chapter will describe the requirements adopted for the approach undertaken and the reasons of why certain parameters are chosen and why others are dismissed.

First of all, as the holographic printing system is multi-view image-based, a determined setting with reference to cameras has to be chosen: it specifies their placement and position in relation to the scene. Four configurations are available: cameras pointing ahead parallel to the projection plane and three sorts of re-centering cameras; two of them are automatically discarded because of distortions introduced. Figure 2.5 displays in a more understandable manner the arrangement of the cameras in those four configurations. The size of the projection plane and its distance to the camera track, or slit, will determine the angle of view and focal length specified in the lens of the camera. Furthermore, the projection plane accommodates the cross-section with the pyramid projected from the

camera, thus making the camera able to image properly the corresponding area.

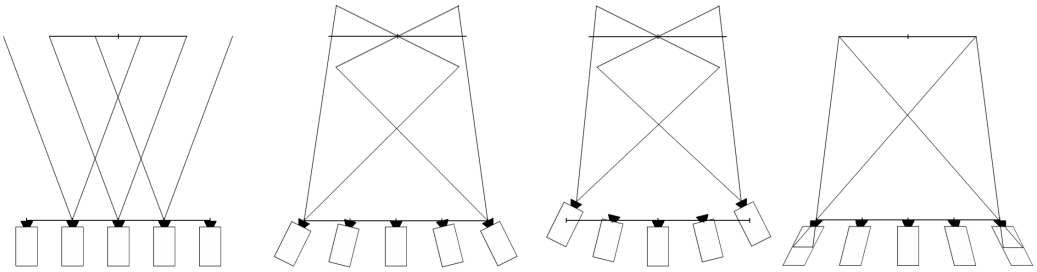


Figure 2.5: On the left, parallel cameras. In the middle, convergent cameras heading the center of the scene along the track and spinning around the center. Theses two configurations lead to distortions. On the right, recentering cameras with their rear films shifted according to the lens to keep the image centered.

Then, the second and third configurations must be discarded. They violate the correspondences in terms of geometry between the position of the cameras and the position of the viewer. The place in the area where images are taken must correspond with the stereogram point of view. The track where the camera moves along must to be straight, remaining with the same geometry as the holographic plane plate. The camera takes view from every position corresponding to the position of the slits. Besides, every point must remain in the same place in the scene regardless where the viewpoint is capturing from. This way, depth is maintained in every view. Complex scenes with too distant and too close points should remain at the same depth from each viewpoint. With a tilted film respecting the projection plane, its contained image would be scaled vertically, known as keystone effect. These two conditions make the two configurations mentioned above incompatible. Furthermore, the EPI obtained, for instance, with the second configuration, lacks of properties that make it useful such as linear tracks.

(a) (b) (c)

Figure 2.6: a) Parallel cameras preserve geometry correspondences, but most of the black region means useless data. b) EPI presents deformed geometry caused by wrong camera configuration. c) Recentered cameras provides useful data regardless the point of view and preserve geometry correspondences.

Regarding the parallel or recentering cameras, the former have a disadvantage. Most of the viewpoints contain image data that could never be seen. Therefore, it is useless

information that is processed but never recorded in the final hologram. Parallel cameras brings an easier setting, but its disadvantage is that it implies too much data out of the projection plane. The fourth configuration showed in Figure 2.5 is considered. A way to achieve these rendered images, as it is explained in the chapter below, consists of using the first configuration and cameras with a wider field of view that accommodate the projection plane even from the most extreme points of views. Figure 2.6 presents the final EPIs that will be produced by the diverse camera configurations.

The parameters chosen for later development are as follows. The resolution of the camera is 600 square pixels and 3 layers for RGB color space. The number of views chosen is 512, what will simplify some post-processing of the renders working with some power of 2. Rendering 512 views from a complex scene might result in a heavy and long computation process. Optimizing the number of cameras through interpolation will be one of the tasks to be polished.

3. Analysis

As it was stated in previous chapters, optimizing the number of cameras involves the analysis of interpolation and re-sampling. With this purpose, the analysis of EPIs has to done in both spacial and frequency domain to understand how to manipulate the images.

3.1 Relations between scene characteristics and Fourier domain representation

Before approaching the study of filtering and interpolation of epipolar images, it may be appropriate a first contact with a property of scenes and its Fourier domain representation, i.e: depth. Depth plays an important role in holographic stereograms. How different depths affect the representation of object in epipolar and frequency domain will be explained in this section.

Initially, one point *z(v,t)* in a particular scene is considered (see Figure 3.1). It is seen by two cameras, *c1* and *c2* with focal length *f*, which go alone the camera track *t*, recording the horizontal position where the two points would fall onto. The two positions where the point *z(v,t)* fall onto the cameras are *v* for *c1* and *v'* for *c2*. The disparity function $v - v' = ft/z$ describes the geometry of the scene.

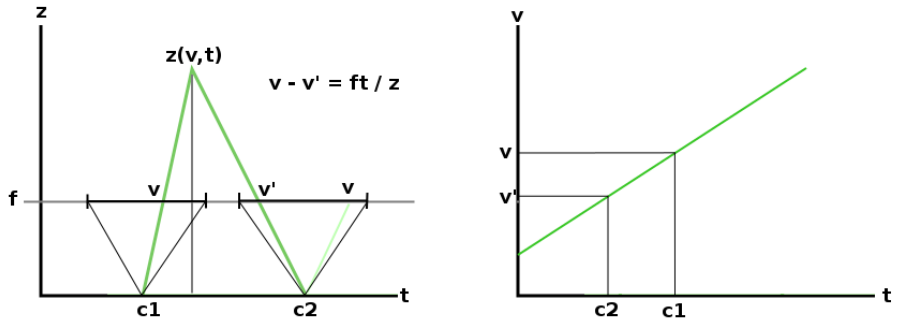


Figure 3.1: On the left, the point placed in the scene captured by both cameras. On the right, line formed by stacking pixel captured along the camera track.

Figure 3.2 depicts the model of a signal with constant depth. The red lines represent

the minimum and maximum depth. The same figure also shows the spectral support of light field. The blue line, with angle of 45º respect to the axis represents the constant depth. The red lines set the spectral support boundaries of depth. The tilt pictured in this domain can be translated into the depth of the signals in the scene.

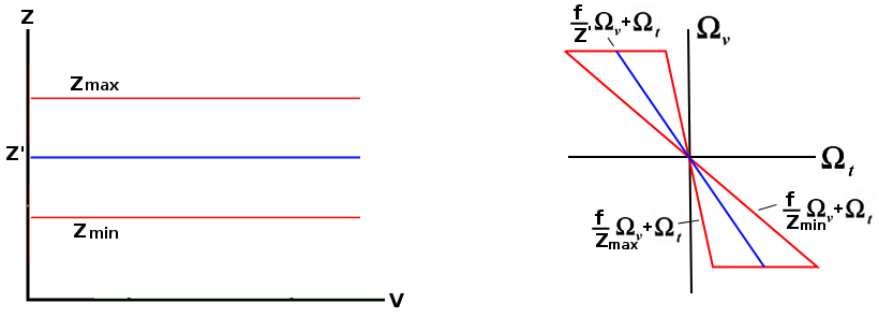


Figure 3.2: On the left, a model of constant depth. On the right, the frequency domain representation with boundaries for minimum and maximum depth.

According to Chai[6], "Any scene with a depth between z_{min} and z_{max} will have its continuous spectral support bounded in the frequency domain". The use of epipolar images connects scene characteristics such as depth with the Fourier domain, which can be adequately manipulated. Finally, a couple of examples of EPIs and their 2D fast Fourier Transform (FFT2D) is shown in Figure 3.3.

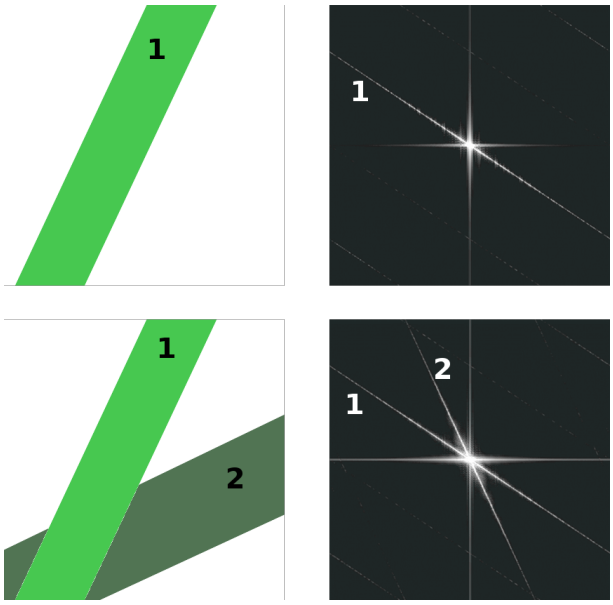


Figure 3.3: The image above displays an example of EPI and its Fourier transform. The image below, with an object behind the first shown above, is further in the scene, and its Fourier transform shows it in its tilt.

3.2 Transform-domain analysis of EPI images

Epipolar plane images reconstructed with all the available views represent the perfect horizontal parallax-only scenario. No information is missing. However, one of the goals of this thesis is to reconstruct the original views with an smaller number of cameras. A straightforward way to generate this scenario is decimation, which directly discards explicit rows. For instance, a decimation of factor 2 would dismiss even rows obtaining a final image with half vertical resolution, a decimation of factor 4 would dismiss three rows for every row that remains and so on. Although this method to manipulate images is clear and it might sound correct initially, it introduces aliasing. Therefore, an anti-aliasing filter was required.

The filter is a low-pass type and it discriminates the high-frequency information from the image. It will produce some visible changes in the plane color regions close to abrupt color changes. Aside from this collateral effect, it softens the sharpness on the borders. The loss of information in the pre-filtering is worth respect to the benefits provided by the posterior interpolations, when applied on anti-aliased images, as the final results will prove.

The filter is applied column-wise given an image (img). With the intention of describing the essence of this technique, a one-dimensional signal (sgn) is considered, representing one column in the image. Figure 3.4 shows this process in more details.

Firstly, the image needs to be enlarged vertically by double of the factor applied, and the first and last value of the signal extended to the beginning and end of the signal correspondingly. The reason of this arrangement resides in the circular property of Fourier transform. If this is not taken into account, the first pixels would interpolate with the last pixels, producing undesirable effects such as mirror at the extremes.

Once the signal sgn is prepared, the Discrete Fourier Transform is applied.

$$sgn\_fourier = DFT(sgn)$$

The signal is now handled in the frequency domain. In this example an interpolation factor of 2 is considered, hence, half of the signal has to be filtered. The filter removes the middle half samples, corresponding to high-frequency. Besides this filtered, the signal has to be normalized, because it should not contain the same energy. The next formulas represent the theory behind this method for this example, not the actual development.

$$normalization = \frac{N.ofcameras}{\frac{N.ofcameras}{2} + 2}$$

$$sgn\_decimated = \frac{sgn\_fourier(1 : \frac{N.ofcameras}{4} + 2, \frac{3}{4}N.ofcameras + 1 : end)}{normalization}$$

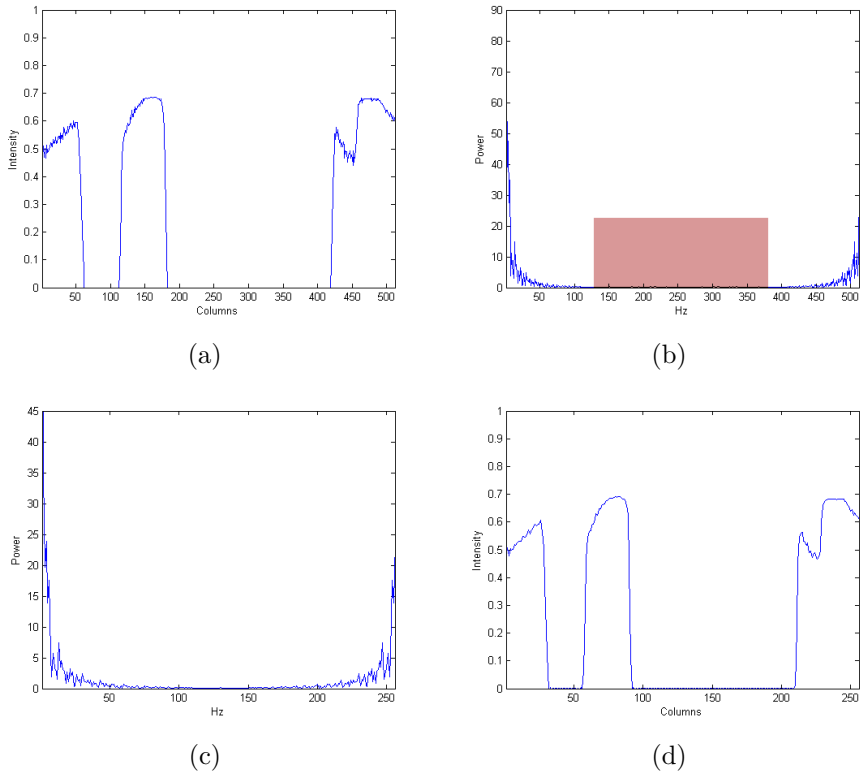


Figure 3.4: a) Original signal of 512 samples. b) Fourier transform of the signal. Highlighted area contains high-frequency region to be filtered. c) Filtered and normalized Fourier transform. d) Final signal, which softened and interpolated after the low-pass filter is applied.

After this step, the signal is filtered in the frequency domain. The final phase include the Inverse Fourier transform, and discard the samples added at the extremes.

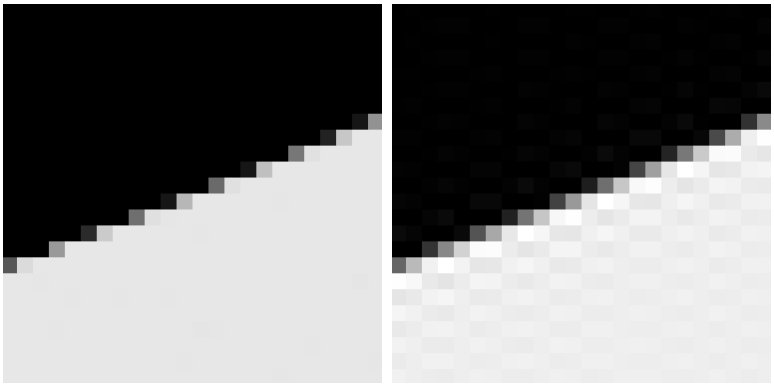
$$sgn\_final\_extended = IDFTsng\_decimated$$

$$sgn\_final = sgn\_final\_extended(2 : end - 2)$$

Figure 3.5 compares two epipolar images with and without pre-filtering. Nevertheless, simple decimation has been applied with the aim of comparing the results, obtained at the point of reconstructing the initially discriminated views.

3.3 Filtering of EPI and optimizing the number of views

At this point of the process, the views are converted into EPIs in five available factors of interpolation. The actual optimization step takes place now, where the pre-filtered



(a) (b)

Figure 3.5: a) Detailed region of an EPI where aliasing is visible. b) The same EPI with anti-aliasing filter.

epipolar images are up-sampled, therefore, dismissed rows which represent missing reconstructed views.

Four of the interpolation used in this phase are typical interpolation kernels available in Matlab: triangle (bilinear), cubic (bicubic), lanczos2 and lanczos3. A new fast dct-based scaling algorithm has been implemented, based on [10].

The next formula describes the interpolation kernel developed. It is applied to 1D signals. When applying this method to images, which are 2D signals, it is as simple as considering every column as an individual signal.

$$signal_k = \sum_{n=0}^{N-1} a_n \{ \quad sincd\left(2N^\circ - 2, 2N, \frac{k+1/2}{factor} - n - 1/2\right) cos\left[\frac{\pi}{2N}\left(\frac{k+1/2}{factor} - n - 1/2\right)\right] + sincd\left(2N^\circ - 2, 2N, \frac{k+1/2}{factor} + n + 1/2\right) cos\left[\frac{\pi}{2N}\left(\frac{k+1/2}{factor} + n + 1/2\right)\right] \}$$

where the term in curly brackets represents an interpolation kernel:

$$N^\circ = min\left(N, \lfloor factor N \rfloor\right)$$

$$sincd\left(M, N, x\right) = \frac{sin\left(\frac{\pi M}{N} x\right)}{N sin\left(\frac{\pi}{N} x\right)}$$

In the previous formulas, N represents the signal length; factor, the interpolation factor chosen; n is the sample processed in every iteration in the summation; and k, means the output sample obtained. The approach has to be applied for every output sample (k) of the signal, in order to calculate its interpolated value. With the help of Matlab these operations can be accomplished easier using vector and matrix operations, specially when it is done in two dimensions.

The previous method has been developed and tested, as it will be shown below in the results chapter, however, it is easy to spot some early limitations. It accepts any interpolation factor greater than 1, but different from (1.4, 4.8, 2.2, 2.6, 3, 3.4, 3.8, 4.2, 4.6, 5, 5.4, 5.8, 6.2) and any factor number odd. For these values, the denominator in *sincd* returns infinite value. This method works quite well in sinusoidal-like signals, but, its results are not that fully acceptable when working with signals with harsh borders as next chapters will show. This is an inconvenient because epipolar images contain signals of this type.

The interpolation methods have up-sampled the epipolar images, but they cannot be directly compared to the original EPIs. The final views have to be reconstructed, which is achieved in the opposite direction to how EPIs are created. Instead of selecting the same row of every view to create an epipolar image, the same row of every EPI is selected, and the new view is obtained. The designing phase will demonstrate how to acquire the reconstructed views from 600 EPIs with 512 pixels of vertical resolution, and then to obtain 512 views with 600 pixels of vertical resolution.

The comparison of original and reconstructed EPIs will depict how accurate the reconstruction is, and consequently, the degree of interpolation that can be applied in relation to the noise that appears. The measurement used to analyse this relation is PSNR (peak signal-to-noise ratio), known as the ratio between the maximum possible power of a signal regarding the power of the noise present in the original and compressed image. Compression understood as the down-sampling and re-sampling executed in epipolar images and translated to scene views.

PSNR contains another measurement method in the algorithm, Mean Squared Error (MSE), that measures the average of the squares of the errors. An original image has a Mean Square Error equal to zero, and therefore, infinite PSNR. The higher PSNR is, the higher quality of the image is.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [Original(i,j) - Interpolated(i,j)]^2$$

where images are mxn size. The final PSNR formula comprises:

$$PSNR = 10log_{10} \left(\frac{MAX^2}{MSE} \right)$$

where MAX is the maximum possible pixel value of the image (255 for images used). The higher the PSNR value is, the better quality of the interpolation is, which is explained further in the last chapter.

4. Design of the multi-view system

This chapter examines the techniques used to obtain necessary elements to build the Computer Generated Hologram system. The following sections will explain the tools developed for obtaining properly the views from a virtual 3D scene, the post-processing performed to them and the re-sampling techniques that achieve the desired optimization in number of views.

Figure 4.1: 1) Rendering of images and cropping to obtain the useful region of it. 2) Creation of full density and pre-filtered EPIs to obtain down-sampled and anti-aliased EPIs. 3) Vertical interpolation of EPIs to recover original resolution. 4) Reconstruction of views from interpolated EPIs.

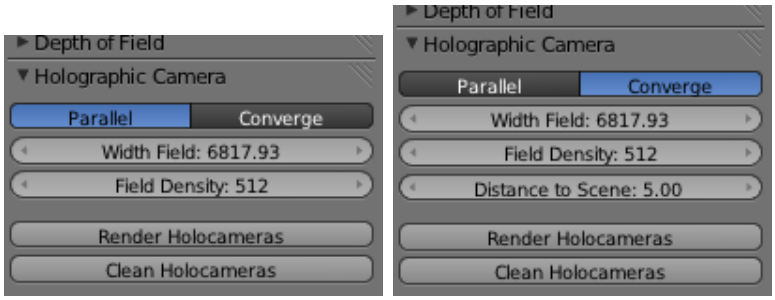
Throughout the developing and testing phases, more than 32.000 files have been created and almost 3GB of data manipulated, which requires a clear and structured folder directory:

```
/.
functions.blend/py/m (1)
scene_name/epis_density (2)
scene_name/epis_density_interpolation (3)
scene_name/parallel_r_density (4)
scene_name/parallel_rc_density (5)
scene_name/parallel_density_interpolation (6)
```

1) The folder represents all the functions and script from Blender, Python and Matlab.
2) The folder contains the full density EPIs and pre-filtered with 5 interpolation factors.
3) Every folder contains the interpolated EPIs with 5 interpolation factors.
4) The folder contains the raw renders.
5) The folder contains the cropped views with proper resolution.
6) The folder contains the reconstructed views for every interpolation factor. In the directory tree presented above, density is represented by a number (512 in this case) and interpolation by one of the 5 interpolation kernel used.

4.1 Building a multi-view system from 3D scenes

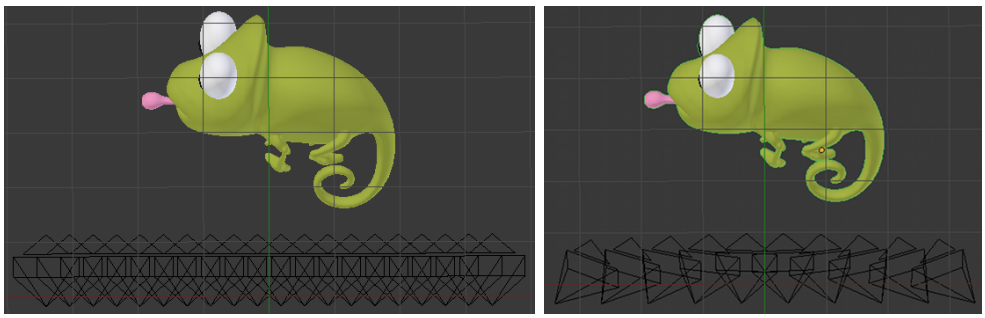
The first step before analysing the views of a scene implies obtaining those views. The software used is Blender as 3D virtual scene and camera setting. The script that places the cameras in the correct place and renders the images is written in Python, and it includes a graphical interface (Figure 4.2).



(a) (b)

Figure 4.2: a) Interface for parallel mode. b) Interface for convergent mode.

Initially, the developed script offered two possible settings, including parallel and convergent cameras (Figure: 4.3). The reason convergent cameras will not provide the proper geometry properties have already been mentioned in previous chapters. Nevertheless, this setting was developed with the aim to obtain graphic proofs of the distortion produced and study its consequences in the epipolar domain.



(a) (b)

Figure 4.3: a) Placement of cameras in parallel mode. b) Placement of cameras in convergent mode.

The attributes that define the scene are linked to the following parameters:

- **Mode:** It switches between parallel and convergent mode.
- **Width Field:** Total length of the camera track (Blender units * 1000).
- **Field Density:** Total number of cameras.

- **Distance to scene (Available only in convergent mode):** Distance to center point of scene where cameras will be focusing on.

According to the parallel mode, the cameras move along the camera track, capturing views parallel to the scene. The camera in the center of the track has the field of view needed for including the whole scene. However, as the camera moves from one side to another, the view captured shows useless information. Figure 4.4 shows how empty data is recorded instead of pixels belonging to the scene.

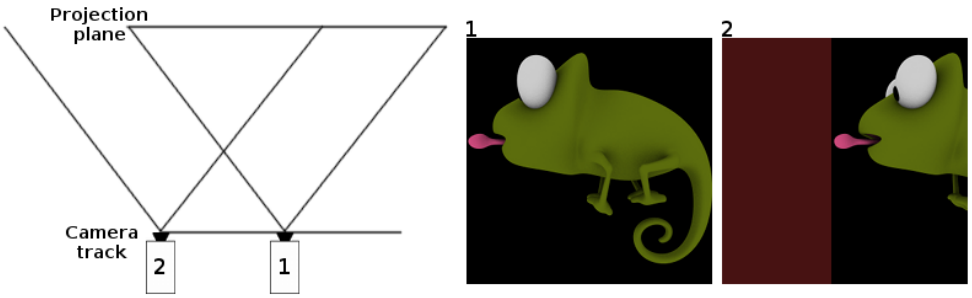


Figure 4.4: The camera number 2, on the extreme of the track, captures useless information (in red).

A simple way to resolve this problem is to enlarge the field of view of the camera, acquiring the entire projection plane from any point of view. This solution uses more rendering information, although the additional rendered regions usually contain none or only a few objects, which does not imply rendering time. Once all views are rendered, it is time to post-process them and convert them into proper views resolution-wise. A script written in Matlab (from now on, all scripts used for image processing will be based on it) crops the image keeping only the region which corresponds with the projection plane. The processed images will have the same resolution after this cropping. Figure 4.5 shows how the following script (crop_renders.m) will transform the raw renders into suitable images for further procedures.

```
function crop_renders(object,density,h_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% h_res: Horizontal resolution of views
% It crops renders to normal resolution
...
for i=1:density
    img = imread(strcat(name_in,int2str(density-i+1)),'png');
    cropped = img(:,(i-1)*2+1:(i-1)*2+h_res,:);
imwrite(cropped,strcat(name_out,int2str(density-i+1),'.png'));
end
```

The scene treated in this work contains 512 views, and the rendering script produces

Figure 4.5: The camera is characterised by a wider field of view. The unnecessary region of the image is dismissed in the image on the right.

images of 1622x600 pixels. Those images are cropped to a final resolution of 600x600 pixels. The following figure (4.6) contains some samples of the output.

Figure 4.6: From left to right, samples of views number 1, 64, 128, 192, 256, 320, 384, 448 and 512.

4.2 Creation and pre-filtering of EPIs

All views obtained in the previous section represent the spatial representation. In order to acquire a light field representation of the scene, the views have to be transformed into epipolar images.

The next steps to follow involve the creation of the mentioned above epipolar images and pre-filtering of those images to remove aliasing. The first script is in charge of creating full density EPIs, which have the same number of rows as the number of views rendered from the scene (512 in this case). A total of different 600 EPIs will be created, as views have 600 pixels of vertical resolution. This is carried out by the next script, which takes the same row of every view to create a single EPI. For example, to create the first EPI, it assigns the first row of the first view to its first row, the first row of the second view to its second row, and so on. It completes this process for 512 rows in the 600 EPIs.

```
function epi(object,density,v_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% v_res: Vertical resolution of views -> Number of EPIs
% It creates epipolar images with full density
```

As a result of this script in this particular scene, the 30-40 first and last EPIs are

completely black, by reason of no objects appearing in the top and bottom rows of the views. Figure 4.7 shows some output samples of this function.

Figure 4.7: From left to right, samples of EPIs number 1, 100, 200, 300, 400, 500 and 600.

As it was explained, the anti-aliasing filter is required before decimation. In the function presented below (a simplified version of the actual one) applies an anti-aliasing filter. It works with 5 different interpolation factors [2, 4, 8, 16, 32] power of 2, which facilitates the processing for Fourier transforms. The down-sampling produces an output shown in the figure 4.8.

```
function prefiltering(object,density,v_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% v_res: Vertical resolution of views -> Number of EPIs
% It filters EPIs with anti-aliasing filter
% and down-samples vertically in 5 factors [2, 4, 8, 16, 32]

%Preparation of extended signal
epi = im2double(...);
epi_ext(1+factor:end-factor,:,:) = epi;
epi_ext(1:factor,:,:) = epi(1,:,:);
epi_ext(end-factor+1:end,:,:) = epi(end,:,:);

%Fourier transform, filtering and normalization
epi_f = fft(epi_ext);
normal = density/((density/factor) + 2);
fraction = density/(2*factor);
epi_fdec = [epi_f(1:fraction+2,:,:); epi_f(end-fraction+1:end,:,:)] / normal;

%Inverse Fourier transform and cropping extended samples.
epi_factor = ifft(epi_fdec);
epi_final = epi_factor(2:end-1,:,:);
```

4.3 Optimizing number of views

At this moment in the design, the following developments will work on the views converted into filtered EPIs to bring them back to their original resolution. The up-sampling that optimizes the number of views necessary to build the multi-view system.

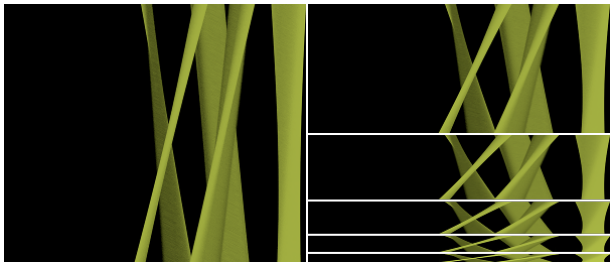


Figure 4.8: The original EPI and the 5 anti-aliased EPIs with 5 different interpolation factors.

The scripts responsible for this step are presented below:

```
function run_epi_interp(object)
% Object: Name of the scene
% It runs epi_interp() with 5 factors [2, 4, 8, 16, 32] and
% 5 interpolation kernels [triangle, cubic, lanczos2, lanczos3, fastdct]

function epi_interp(object,density,factor,v_res,interpolation)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% Factor: Interpolation magnitude
% v_res: Vertical resolution of views -> Number of EPIs
% interpolation: Interpolation kernel used
% It up-samples vertically EPIs
```

The function epi_interp uses either imresize() Matlab function for bilinear, bicubic, lanczos2 and lanczos3 kernels interpolation, or the next function fdctEPI, which implements the fast DCT-based scaling algorithm [10].

This algorithm was implemented to scaling 1D signals, and later it was adapted to two dimensions. All the scripts coded that improved that code time-wise in every version are shown in the appendices. The next sample of code (simplified version) aims to demonstrate and present the basics of the fast DCT interpolation.

```
function img_out = fdctEPI(img_in,factor)
% img_in: input image
% factor: interpolation magnitud
% It up-samples vertically img_in using fastDCT interpolation

No = rows_in;
n = repmat(0:rows_in-1,rows_out,1);
k = repmat((0:rows_out-1)',1,rows_in);

DCT = sincd(2*No-2, 2*rows_in, (k+1/2)/factor -n -1/2) .* ...
```

```
cos( pi/(2*rows_in) * ( (k+1/2)/factor -n -1/2) ) + ...
sincd((2*No-2), 2*rows_in, (k+1/2)/factor +n +1/2) .* ...
cos( pi/(2*rows_in) * ( (k+1/2)/factor +n +1/2) );

% Multiplication by DCT-Matrix to all image component layers
for i=1:components
img_out(:,:,i) = DCT*img_in(:,:,i);
end

% Truncating negative and greater than 1 values
img_out(img_out<0) = 0;
img_out(img_out>1) = 1;

function result = sincd(M,N,x)
% sincd(M,N,x) = sin(piMx/N)/(Nsin(pix/N))
% Digital sinc function used in fastDCT
```

One of the critic code sections in fdctEPI() is the population of n and k, as they represent the matrices which substitute the initial loops, operated by rows and columns. Executing this operation with two nested loops makes its computation inefficient. In a preliminary version one of the loops was substituted by a vectorized operation, but, populating two matrices and carrying linear operation is more coherent.

Matrix k and n respond to the following pattern:

$$k = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ n & n & \cdots & n \end{pmatrix} n = \begin{pmatrix} 0 & 1 & \cdots & n \\ 0 & 1 & \cdots & n \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & n \end{pmatrix}$$

Three methods of populating matrices might be considered.

```
% Kronecker tensor product
ns = kron([1:size_in]-1,ones(1,size_out)');
ks = kron(ones(1,size_in),[1:size_out]'-1);

% Replicating matrices
ns = repmat([0:size_in-1],size_out,1);
ks = repmat([0:size_out-1]',1,size_in);

% Manual with ones
temp= [0:size_in-1];
ns= temp(ones(1,size_out),:);
temp= [0:size_out-1]';
ks= temp(:,ones(1,size_in));
```

The results for resolution 300, 600 and 900 pixels and factors 2-10 are displayed in Figure 4.9.

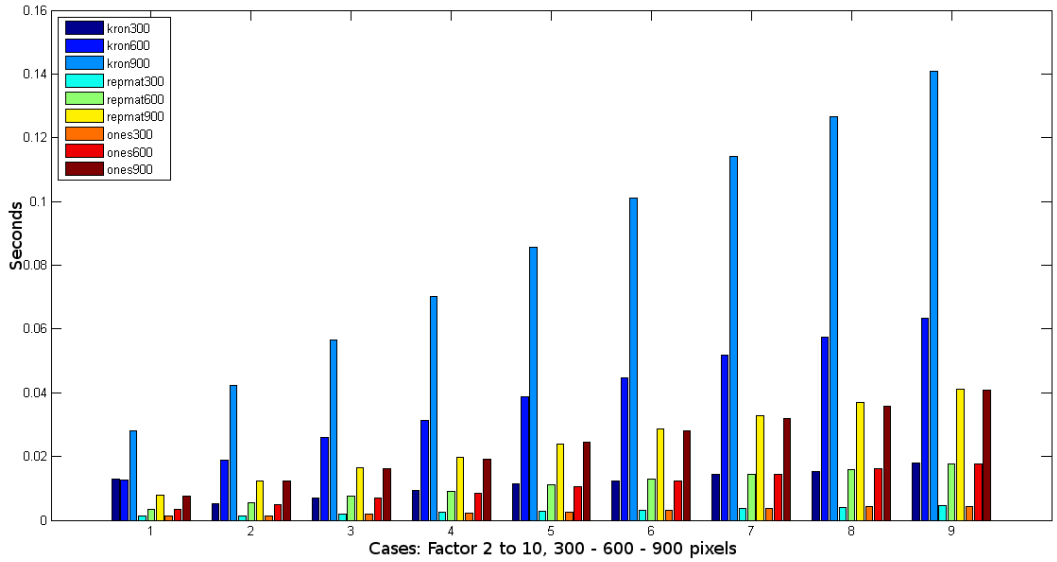


Figure 4.9: Kron method is notably worse than the other two. There is no difference between repmat and manual, at the same time, manual requires less memory and number of instructions.

The reconstruction of the views from the interpolated EPIs using the kernels explained above is straightforward. A view is formed by stacking rows coming from the same row of every EPI. The following functions are responsible for the final phase.

```
function run_reconstruct_views(object)
% object: Name of the scene
% It runs reconstruct_views() with 5 factors [2, 4, 8, 16, 32] and
% 5 interpolation kernels [triangle, cubic, lanczos2, lanczos3, fastdct]

function reconstruct_views(object,density,factor,v_res,interpolation)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% Factor: Factor of interpolation applied vertically
% v_res: Vertical resolution of views -> Number of EPIs
% interpolation: Method/kernel used for re-sampling
% It reconstructs all the views from the re-sampled EPIs
```

The whole process seen from a graphical perspective is pictured in Figure 4.10.

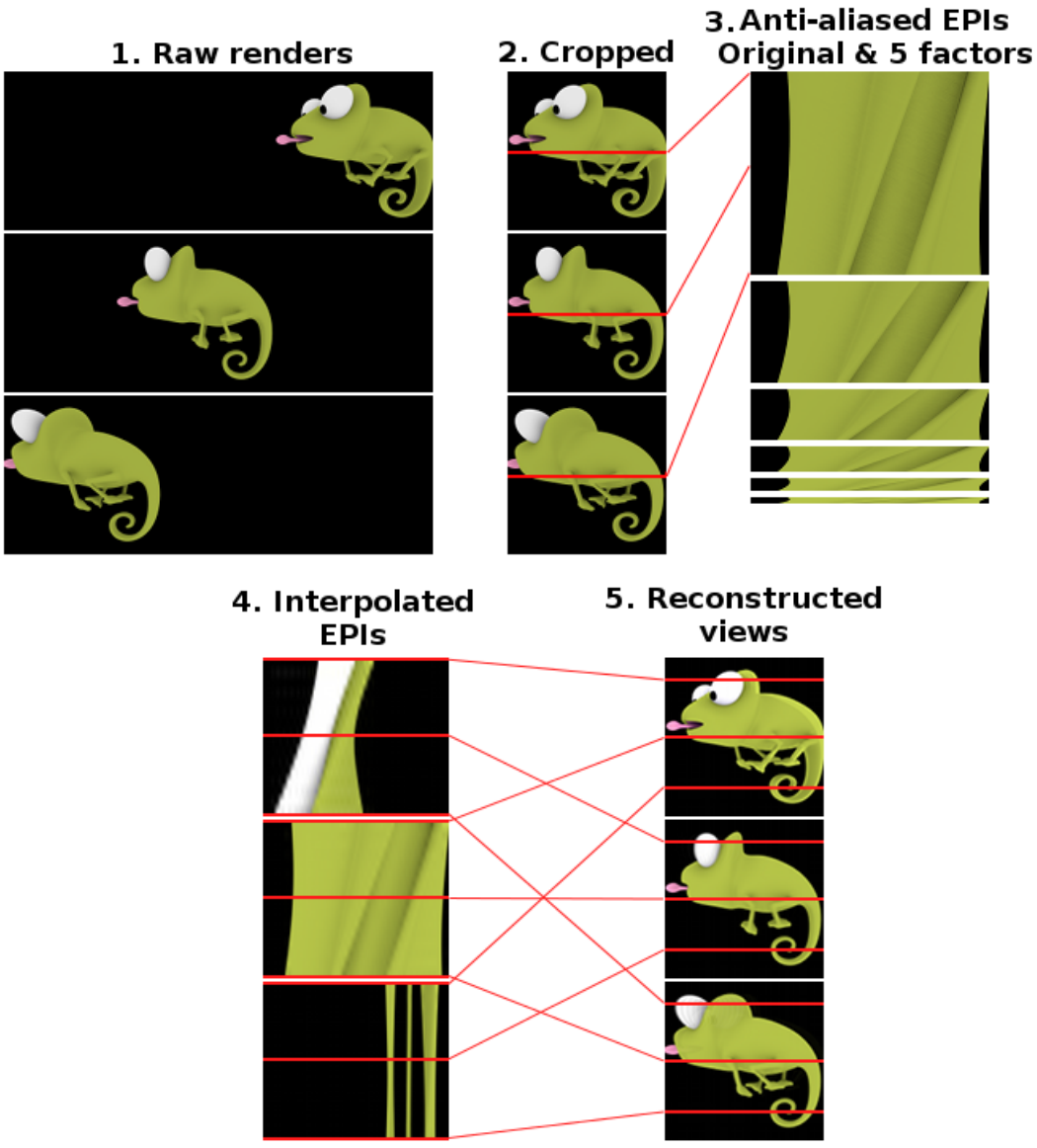


Figure 4.10: The figure shows all the steps taken in this work, and how images are transformed and manipulated in every step.

5. Results

In this chapter all graphic and numeric results will be presented and interpreted. The main tests executed in this work involved the fast DCT algorithm [10] and other interpolation kernels, as well as the final comparison tests between original and interpolated views.

5.1 Fast DCT algorithm

Fast DCT scaling algorithm has been used to up-sample vertically epipolar planar images. Before the image tests, a set of peculiar one-dimensional signals was tested towards a better understanding of its effects in the particular cases. The next figure (5.1) shows the effects of scaling in some peculiar signals and simple images based on those signals.

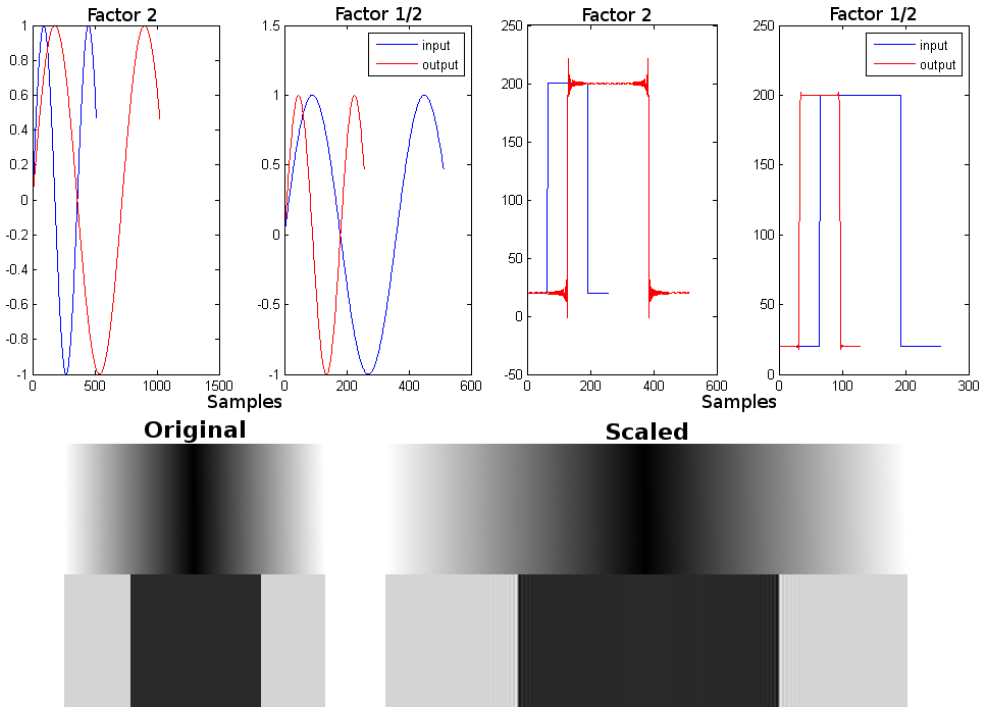


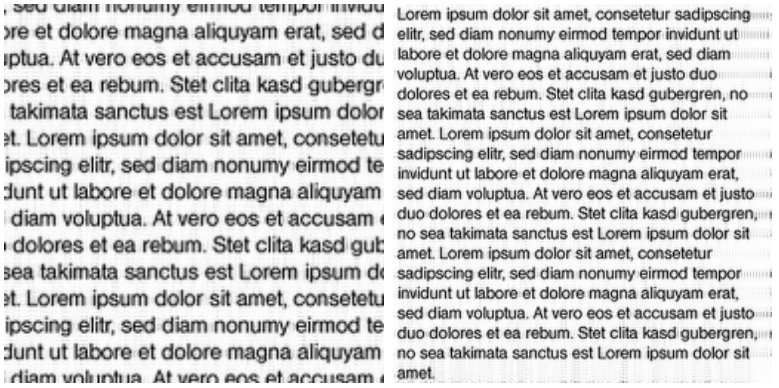
Figure 5.1: Scaling produced over the sharp signal causes sinusoidal effect.

Consequence of this effect, the borders of epipolar images, which contain basically lines and stripes, present blurring. Another example, tested to confront the outcome of the paper [10] against this work, presents iteratively scaling of a text image. The results throw the impression that regardless how many iterations are executed, the text remains readable unlike with other interpolation kernels. Figure 5.2 show the difference between interpolation kernels clearly.

(a) (b) (c) (d)

Figure 5.2: a) Original text image. 75 iterative zoom-in&zoom-out scaling, factor sqrt(2) b) by bilinear algorithm, c) by bicubic algorithm, d) by Fast DCT algorithm.

As it was mentioned above, the border artefacts present will have to be tackled down in posterior steps. The artefacts appearing beyond the borders will take its utility away. Figure 5.3 depicts why the displayed results are not useful for epipolar image interpolation.



(a) (b)

Figure 5.3: Text image scaled 1000 times with factor sqrt(2). a) Region of output text image displayed alike [10]. b) Complete text image, contains the artifacts on the borders.

Another aspect considered in this text is the computing time spent by all these methods. The results presented in Figure 5.4 prove that the implementation of Fast DCT coded in Matlab is slower than built-in functions Matlab, since fast DCT is comprised of two external functions. The hardware used in the execution of these tests does not influence the results, as we are not comparing time-wise data with any other results obtained in other hardware. Furthermore, every test was executed 10 times in the same conditions. Figure 5.4 show the average of all the executions.

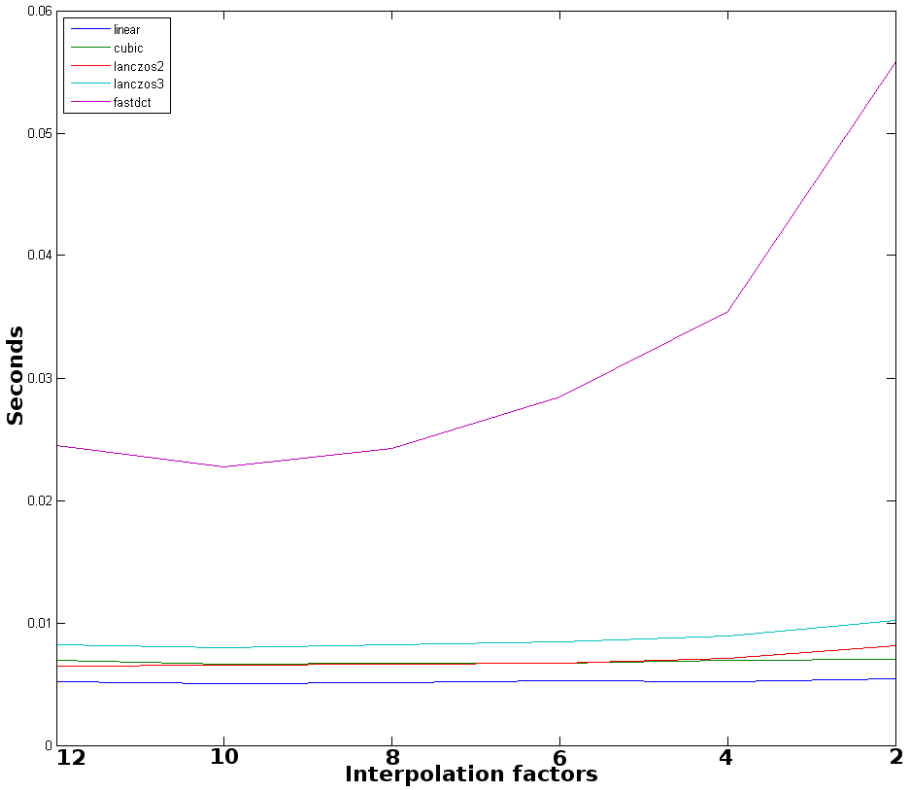


Figure 5.4: The implementation of the fast DCT algorithm is considerably slower than built-in algorithms.

5.2 Experimental results

The most important result to be discussed involves original and interpolated EPIs. This comparison was accomplished through PSNR (Peak Signal-to-Noise Ratio) measurement, as explained in the analysis chapter.

The first experimental result with PSNR checks every reconstructed view for every interpolation method. The following script was written to obtain the results:

```
function test_psnr(object,density,factor)
% Object: Name of the scene
% Density: Number of views
% Factor: Interpolation magnitud
% It calculates the PSNR for every factor and view
```

The visible peaks that appear in Figure 5.5 represent the true views, non-interpolated remaining in the filtered EPI. A view closer to a true view will get, therefore, a better PSNR result. According to this, the extreme views might show the worst values, since they do not have neighbour views to interpolate with. Figure 5.5 belongs to the test with factor 8. The tests with the rest of interpolation factors are available in the appendices.

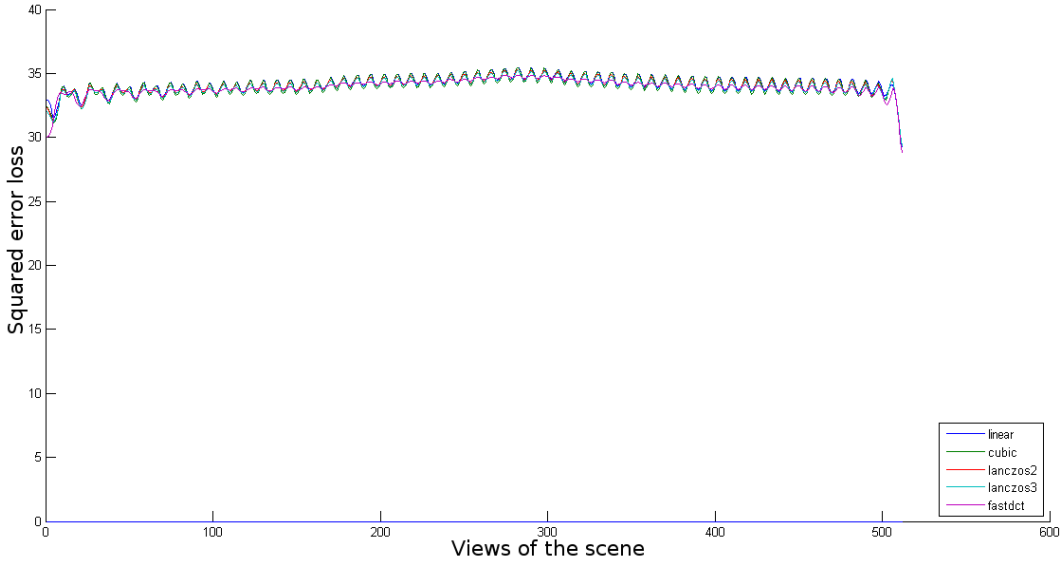


Figure 5.5: Peaks represent original and therefore non-interpolated views. Views on the extreme tend to generate worse result because of their isolation.

Also, another script was written to compare all the interpolation factors in the same figure. The complete script is available in the appendix:

```
function PSNR = test_psnr_average(object,density)
% Object: Name of the scene
% Density: Number of views
% Factor: Interpolation magnitud
% It calculates the PSNR for every factor and its average between all the
% views
```

The same graph is used to contrast the PSNR values obtained in the tests with pre-filtered EPIs confronted to the same test with only decimated EPIs. In the case of views based on aliased epipolar images, the maximum factor applied was 12, since it was expected that greater factors would provide PSNR values utterly unacceptable.

Pre-filtered EPIs-based views are presented as points, and non-pre-filtered as lines. The first aspect to notice is that the difference between interpolation kernels is not sufficiente in such magnitude, hence, the interpolation factor will be the decisive parameter.

Unlike it could have seemed in the analysis of Fourier domain of EPIs, dismissing certain information related to high-frequency, it is proved now that an anti-alising filter is sufficient useful. In the next chapter, this graph 5.6 will be discussed further. The minimum number of cameras needed to reconstruct an acceptable multi-view system will be determined accordingly to this last figure.

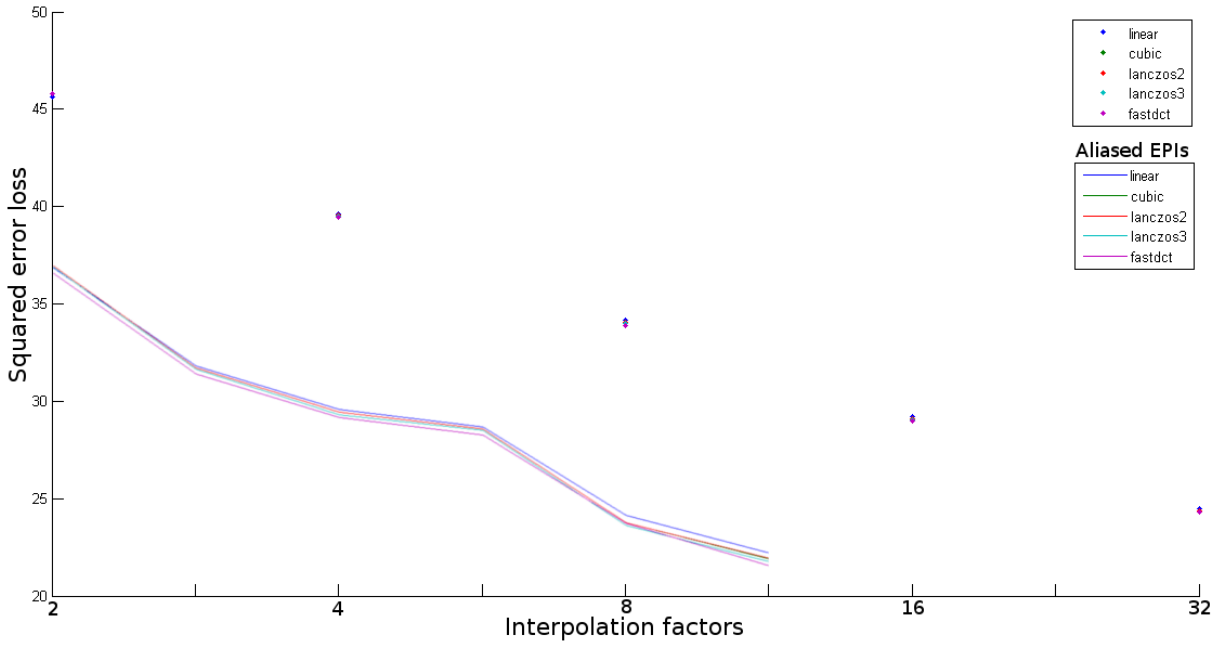


Figure 5.6: Views based on pre-filtered EPIs obtain much better results for the same interpolation factor than views based on simple decimated EPIs.

6. Conclusions

In this work, the goal of designing the tools are covered as well as the techniques that make possible a multi-view system based on a virtual possible scene. Besides, the characteristics of the scene have been analysed in the spatial and frequency domain. Comprehending these properties allow us to develop new methods and interpret the result produced during the tests. Finally, a set of tests has presented the efficiency of every method and its limitations.

6.1 Conclusions about the results

A new filter based on [10] has been developed. It has been proved that the filter works better in overall for interpolating text images iteratively. However, in certain cases with sharp borders, like epipolar planar images, the artifacts produced around the borders affect the overall performance at the moment of view reconstruction.

The fast DCT algorithm has been implemented in Matlab and therefore it is slower than the built-in functions.

The optimization of the cameras has been done through vertical interpolation of images. These Epipolar Planar Images represent how the light field of every pixel changes along different points of view. In order to obtain better results after the up-sampling, an anti-aliasing filter has been applied. The results displayed in the previous chapter prove that filtering EPIs benefit the overall outcome.

The tests executed to determine the quality of the reconstructed views are based on Peak Signal-to-Noise Ratio measurement. The images compared were represented in RGB. Another option is to convert these images first into YCbCr color space, and compare only the Y layer (luminance). The reason resides in the higher capability of the human visual system to perceive disturbances and noise in brightness than its sensitiveness to colour.

Finally with PSNR experimental tests, it is possible to make an estimation of the optimum number of cameras. Figure 6.1 helps to understand this estimation. There is no exact value of proper number of cameras to choose, yet an interval where the results will be acceptable with reference to the intention of application. PSNR does not dictate

a final decision. Some visual results, perceived for the human visual system better, might lead to worse PSNR values. Interpolations are also highly-dependent of the scene chosen. In this work, the virtual chameleon object contains many different shapes and surfaces, and it is middle depth placed.

Nevertheless, the possibility to try the multi-view system developed along with the virtual scene interpolated and optimized in a hologram plate is not possible. Therefore, the visual looking is not considered, and the judgement taken into account is the PSNR measurement. An acceptable value for lossy images lays between 30dB and 50dB. A band is overlapped in the figure 6.1. A great difference in performance between interpolation methods is not found. As a direct consequence, choice of the fastest and simplest method indicates the recommended option. Triangle interpolation kernel, also known as bilinear, is the fastest and rather provides the best PSNR values not distant from the rest.

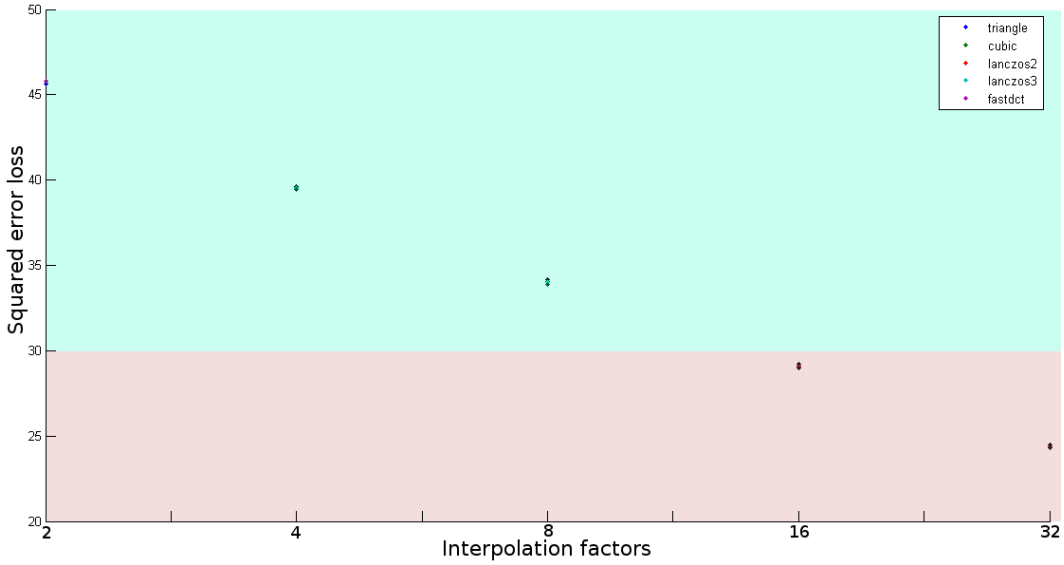


Figure 6.1: The blue band comprises very good PSNR values for lossy images. The red band comprises lossy images for wireless transmission beneath our objective sort of quality required.

Consequently, the factors of interpolation which provide good quality multi-view systems belong to the interval [2-16], in other words, using from one half to one sixteenth of the initial number of views. 512 views were used in this work. 32 views processed with anti-aliasing and re-sampling algorithms allow to reconstruct the initial 512 views with 29dB of square error loss in the PSNR test. Using an interpolation factor of 32 would generate a relative good PSNR result (around 24); however, it needs to be coherent with some criteria and boundaries set.

6.2 Future Work

This thesis studies an initial attempt to establish a methodology for a multi-view rendering. It uses virtual cameras due to their freely manipulation of parameters. It also takes advantages of the 3D environment to render objects and scenes that can be adapted in a better way to the requirements comparing to real-life objects.

The next step involves the use of physical cameras to acquire views, new interpolation techniques including mixed techniques. Diverse advanced approaches respect to the camera setting, more complex re-centered cameras that lead to a full-parallax setting. Also intrinsic parameters of lenses in real cameras will play an important role in depth-related topics.

Bringing the study to possible application means to be the next step to take. For instance, computer gaming graphics with real-time interaction require more and more of simplified polygon models, they take advantage of advanced texture mappings instead. The simpler models can be better manipulated in the epipolar domain, and the quality of the final product will be more dependent on the texture level.

The work of this project symbolises a number of independent steps in order to achieve a final result. However, a future research may be developing all the steps in a single unit of production. The specifications of such production software would vary considerably, regarding the different platforms, tools and developing software used during the process. If a production tool has to be developed, some aspect should be taken into account. For example, all the algorithms shall be written in compiled languages, where optimization could occur. Mex or C might be a starting point as an alternative for Matlab. Some parts of the code which require a huge input/output transfer can be parallelized. The most important aspect in the optimization of code resides in the Amdahl's Law: the optimization of a block of code will optimize the overall application in the same proportion as that code represent the overall process. Sometimes a lot of effort is put into some optimization that, as it happened in the study of matrix population, it does not reflect a worth benefit in comparison to the time spent.

6.3 Personal Assessment

This thesis has meant an utter special project in the period that has happened, for many reasons. Firstly, I faced this project in an international environment, as part of my Erasmus stay in Finland at Tampere university of Technology. Secondly, being a researcher in the department of Signal Processing made me discover a side in the university that I have not been involved before. It also gave me the possibility to attend the 3D Media Training School in the summer of 2012 in Tampere.Finally, although my background is entirely based on Computer Science, the challenge to adapt my knowledge and comprehend algorithms and properties related to signal processing (most of them not studied

in my degree before) has enforced me to raise the problems from a different perspective of abstraction. Despite, making use of my software and computer expertise has been of help when developing certain algorithms. My vision of how images are processed and the technology used behind has changed after this thesis has been completed.

Bibliography

[1] E. N. Leith. *White light holograms*. Scientific American, (1976).

[2] Michal W. Halle. *The Generalized Holographic Stereogram*. Massachusetts Institute of Technology, (1991).

[3] Michal W. Halle, Adam B. Kropp. *Fast computer graphics rendering for full parallax spatial displays*. Brigham and Women's Hospital, Massachusetts Institute of Technology, (1997).

[4] Ravikanth Pappu *et al. A Generalized Pipeline for Preview and Rendering of Synthetic Holograms*. Massachusetts Institute of Technology, Interval Research Corporation, (1997).

[5] Michal W. Halle. *Multiple Viewpoint Rendering*. Brigham and Women's Hospital, (1998).

[6] Jin-Xiang Chai *et al. Plenoptic Sampling*. Microsoft Research China, (2000).

[7] Wendy Plesniak *et al. Reconfigurable Image Projection Holograms*. Brigham and Women's Hospital, MIT Media Laboratory, ThingMagic, Inc., (2006).

[8] Smithwick, Quinn Y. J. *et al. Real-time Shader Rendering of Holographic Stereograms*. Society of Photo-Optical Instrumentation Engineers, (2009).

[9] Melania Paturzo *et al. Holographic Display of synthetic 3D dynamic scene*. 3D Research Center and Springer, (2010).

[10] Leonid Yaroslavsky, Leonid Bilevich. *Fast DCT-based Algorithm for Signal and Accurate Scaling* . Tel Aviv University, (2012).

7.2 Metodología de trabajo

La metodología utilizada en este proyecto se caracteriza por circunstancias particulares. Previa a la presentación del proyecto como una MSc Thesis, se solicitó el diseño de una versión preliminar de la aplicación Blender, la cual renderiza la escena desde diferentes puntos de vista, como una prueba de conocimientos de programación para continuar con el resto del desarrollo. Más adelante, después del análisis de trabajos anteriores, cada fase fue analizada, diseñada, implementada y finalmente testada. Si los tests muestran que los requisitos se han cumplido, la fase de desarrollo se considera finalizada. La escritura de la memoria comenzó simultáneamente con las fases de diseño y testado.

7.3 Diagrama de Gantt

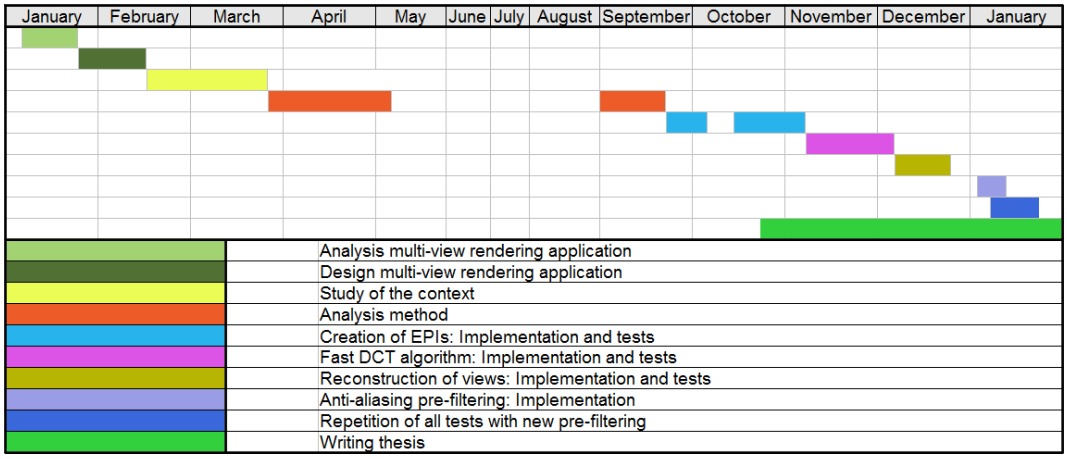


Figura 7.1: Diagrama de Gantt.

El diagrama muestra un periodo en verano donde el proyecto estuvo paralizado. La razón es un trabajo de verano en Tampere, Finlandia, que me permitió extender mi estancia en este país desde Septiembre en adelante, cuando las ayudas económicas de la beca Erasmus habían finalizado.

7.4 Herramientas utilizadas

A lo largo del análisis, diseño y escritura de la tesis se utilizaron varias herramientas. Blender es una plataforma de código abierto para gráficos por ordenador 3D, utilizada para configurar el escenario con la escena virtual y cámaras, así como herramienta de rendering. Los scripts escritos en Blender están basados en Python. La herramienta principal usada a lo largo del proyecto ha sido Matlab, para el procesado de imágenes y

el desarrollo y testado de algoritmos. En la escritura de la presente memoria, el kernel LaTeX y el editor TeX Maker han sido los elegidos, junto con GIMP para manipular las imágenes presentadas en este trabajo.

7.5 Código - Sistema de rendering multi-vista

Fichero holographic_camera_1_2_0.py

```
#  Author: David Valle
#  Departament of Signal Processing (SGN)
#  David Valle (230113) david.valle@tut.fi
#

bl_info = {
    'name': "Holographic Camera",
    'author': "David Valle <david.valle@tut.fi>",
    'version': (1, 0, 0),
    'blender': (2, 6, 0),
    'api': 41098,
    'location': "Select a Camera > Properties Panel > Camera Panel >
                                                    Holographic Camera",
    'description': "Render images for a future holographic model",
    'warning': "",
    'wiki_url': "",
    'tracker_url': "",
    'category': "Object"}

import bpy
import mathutils
from math import *
from bpy.props import *

# GUI (Panel)
class OBJECT_PT_holographic_camera(bpy.types.Panel):

    bl_label = "Holographic Camera"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "data"

    # show this add-on only in the Camera-Data-Panel
    @classmethod
```

```
def poll(self, context):
    return context.active_object.type  == 'CAMERA'

bpy.types.Object.holographic_width_field = bpy.props.FloatProperty(
    attr="holographic_width_field",
    name='holographic_width_field',
    description='Camera Width Field',
    min=0.0, soft_min=0.0, max=1000000, soft_max=1000000, default=100)

bpy.types.Object.holographic_field_density = bpy.props.IntProperty(
    attr="holographic_field_density",
    name='holographic_field_density',
    description='Camera Field Density',
    min=0, soft_min=0, max=1000000, soft_max=1000000, default=5)

bpy.types.Object.holographic_distance_to_scene = bpy.props.FloatProperty(
    attr="holographic_distance_to_scene",
    name='holographic_distance_to_scene',
    description='Distance to Scene',
    min=0.000, soft_min=0.000, max=1000000, soft_max=1000000, default=300)

bpy.types.Object.show_mesh_distance = bpy.props.BoolProperty(
    attr="show_mesh_distance",
    name="show_mesh_distance",
    default=False)

bpy.types.Object.holocamera_type = bpy.props.EnumProperty(
    attr="holocamera_type",
    items=( ("PARALLEL", "Parallel", "Parallel camera distribution"),
            ("CONVERGE", "Converge", "Camera distribution with convergency")),
    name="holocamera_type",
    description="",
    default="PARALLEL")

# Draw
def draw(self, context):
    layout = self.layout

    holocamera = context.scene.camera
    row = layout.row()
    row.prop(holocamera, "holocamera_type", text="Holographic Camera Type",
                                                          expand=True)
    row = layout.row()
    row.prop(holocamera, "holographic_width_field", text="Width Field")
```

```
        row = layout.row()
        row.prop(holocamera, "holographic_field_density", text="Field Density")

        if holocamera.holocamera_type == "CONVERGE":
            row = layout.row()
            row.prop(holocamera, "holographic_distance_to_scene",
                                        text="Distance to Scene")

        if holocamera.holocamera_type == "PARALLEL":
            pass

        row = layout.row()
        row.separator()

        row = layout.row()
        row.operator('holocamera.render_holographic_cameras')
        row = layout.row()
        row.operator('holocamera.clean_holographic_cameras')

# Operator 'Render Holographic Cameras'
class OBJECT_OT_render_holographic_cameras(bpy.types.Operator):
    bl_label = 'Render Holocameras'
    bl_idname = 'holocamera.render_holographic_cameras'
    bl_description = 'Render the Holographic Cameras'
    bl_options = {'REGISTER', 'UNDO'}

    #Call the operator 'Render Holographic Cameras'
    def execute(self, context):

        cam = bpy.data.objects["Camera"]
        density = cam.holographic_field_density
        width = (cam.holographic_width_field/1000)/(density-1)

        if cam.holocamera_type == "CONVERGE" :
            #Distance to focus, 1/100 blender units
            distance= cam.holographic_distance_to_scene

        #Save original location of camera to restore it at the end
        orig_loc = cam.location.copy()

        #Save original rotation of camera to restore it at the end
        orig_rot = cam.rotation_euler.copy()

        local_pos = cam.location.copy()
```

```
local_rot = cam.rotation_euler.copy()
print('local_rot(x,y,z)= '+str(local_rot))

#Hint: Way to bring local coord. to world coord. (world = local*matrix)
matrix = bpy.data.objects['Camera'].matrix_world.copy()


sceneKey = bpy.data.scenes.keys()[0]
scene = bpy.data.scenes[sceneKey]
scene.camera = cam

for object in scene.objects:
    if object.name[:7] == "Camera.":
        scene.objects.unlink(object)

i = floor(density / 2)
if 0 == 0 :
#for i in range(density):
    for j in range(density):

        #Currently location of camera
        print()
        print('('+str(i)+','+str(j)+')')
        local_pos.x = (j - (density-1)/2)*width
        local_pos.y = ((density-1)/2 - i)*width
        cam.location = matrix*local_pos.copy()
        print('local_pos(x,y,z)= '+str(local_pos))
        print('world_pos(x,y,z)= '+str(cam.location))

        #Rotate cameras in converge method
        if cam.holocamera_type == "CONVERGE" :

            #Currently rotation of camera
            cam.rotation_euler.x -=
                        atan( ((density - 1) / 2 - i)*width / distance)
            cam.rotation_euler.y -=
                        atan( ((density - 1) / 2 - j)*width / distance)
            cam.rotation_euler.z = 0.0

            #Set output file path
            #scene.render.filepath =
                         '//chameleon/converge_600/'+str(i)+'_'+str(j)
            scene.render.filepath = '//chameleon/converge_600/'+str(j)
        else:
```

```
                    #Parallel
                    #scene.render.filepath =
                             '//chameleon/parallel_300/'+str(i)+'_'+str(j)
                    scene.render.filepath =
                             '//chameleon/parallel_r_512/'+str(j+1)

                #Create multiple cameras
                #bpy.ops.object.camera_add(location=(cam.location.x,
                cam.location.y, cam.location.z),rotation=(cam.rotation_euler.x,
                cam.rotation_euler.y, cam.rotation_euler.z))

                #Render camera
                bpy.ops.render.render( write_still=True )

                #Restoring initial position and rotation of camera
                cam.location = orig_loc.copy()
                cam.rotation_euler = orig_rot.copy()

        cam = bpy.data.objects["Camera"]
        bpy.context.scene.objects.active = cam

        return {'FINISHED'}



# Operator 'Render Holographic Cameras'
class OBJECT_OT_render_holographic_cameras(bpy.types.Operator):
    bl_label = 'Clean Holocameras'
    bl_idname = 'holocamera.clean_holographic_cameras'
    bl_description = 'Clean the Holographic Cameras'
    bl_options = {'REGISTER', 'UNDO'}

    #Call the operator 'Render Holographic Cameras'
    def execute(self, context):

        sceneKey = bpy.data.scenes.keys()[0]
        scene = bpy.data.scenes[sceneKey]

        for object in scene.objects:
            if object.name[:7] == "Camera.":
                scene.objects.unlink(object)

        cam = bpy.data.objects["Camera"]
        bpy.context.scene.objects.active= cam
```

```
        return {'FINISHED'}

# Register
def register():
    bpy.utils.register_module(__name__)


def unregister():
    bpy.utils.unregister_module(__name__)

if __name__ == "__main__":
    register()
```

7.6 Código - Recortado de renders

Fichero crop_renders.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi

function crop_renders(object,density,h_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% h_res: Horizontal resolution of views
% It crops renders to normal resolution

     name_in = strcat(object,'/parallel_r_',int2str(density),'/');
     name_out = strcat(object,'/parallel_rc_',int2str(density),'/');

    for i=1:density
         i
        img = imread(strcat(name_in,int2str(density-i+1)),'png');
        cropped = img(:,(i-1)*2+1:(i-1)*2+h_res,:);
        imwrite(cropped,strcat(name_out,int2str(density-i+1),'.png'));
    end
end
```

7.7 Código - Creación de EPIs con densidad completa

Fichero epi.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Creation of EPI's (full density)

function epi(object,density,v_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% v_res: Vertical resolution of views -> Number of EPIs
% It creates epipolar images with full density

    % Concatenation of input/output path
     name_in = strcat(object,'/parallel_rc_',int2str(density),'/');
     name_out = strcat(object,'/epis_',int2str(density),'/');

    % Construction of EPI, one per row, same amount as vertical
    % resolution of views
    for i=1:v_res
        i
        for j=1:density
            img = imread(strcat(name_in,int2str(j)),'png');
            img_epi(j,:,:) = img(i,:,:);
        end
        imwrite(img_epi,strcat(name_out,int2str(i),'.png'));
    end
end
```

7.8 Código - Pre-filtrado de EPIs

Fichero prefiltering.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Anti-alising pre-filter to EPIs.
% Output: EPIs prefiltered with different densities.

function prefiltering(object,density,v_res)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% v_res: Vertical resolution of views -> Number of EPIs
% It filters EPIs with anti-aliasing filter
% and down-samples vertically in 5 factors [2, 4, 8, 16, 32]

    % Concatenation of input/output path
    name = strcat(object,'/epis_',int2str(density),'/');

    for i=1:v_res
        epi = im2double(imread(strcat(name,int2str(i)),'png'));
        for j=1:5
            factor= pow2(j);

            %Preparation of extended signal
            epi_ext = zeros(factor+density+factor,v_res,3);
            epi_ext(1+factor:end-factor,:,:) = epi;
            for z=1:factor
                epi_ext(z,:,:) = epi(1,:,:);
                epi_ext(end-z+1,:,:) = epi(end,:,:);
            end

            %Fourier transform, filtering and normalization
            epi_f = fft(epi_ext);
            normalization= density/((density/factor) + 2);
            fraction=density/(2*factor);
            epi_fdec = [epi_f(1:fraction+2,:,:);
                        epi_f(end-fraction+1:end,:,:)] / normalization;
            epi_fdec(fraction+2,:,:) = real(epi_fdec(fraction+2,:,:))*2;

            %Inverse Fourier transform and cropping extended samples.
            epi_factor = ifft(epi_fdec);
```

```
            epi_final = epi_factor(2:end-1,:,:);

            imwrite(epi_final,
                    strcat(name,int2str(i),'_',int2str(factor),'.png'));
        end
    end
end
```

7.9 Código - Interpolación de EPIs

Fichero epi_interp.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Interpolation of EPI's

function epi_interp(object,density,factor,v_res,interpolation)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% Factor: Interpolation magnitud
% v_res: Vertical resolution of views -> Number of EPIs
% interpolation: Interpolation kernel used
% It up-samples vertically EPIs

    % Concatenation of input/output path
    name_in = strcat(object,'/epis_',int2str(density),'/');
    name_out = strcat(object,'/epis_',int2str(density),'_',interpolation,'/');

    for i=1:v_res
        epi = imread(strcat(name_in,int2str(i),'_',int2str(factor)),'png');

        if strcmp(interpolation,'fastdct') % Use of fast dct interpolation
            interp = fdctEPI(epi,factor);
        else
            interp = imresize(epi, [density size(epi,2)], interpolation);
        end
        imwrite(interp,strcat(name_out,int2str(i),'_',int2str(factor),'.png'));
    end
end
```

Fichero fdctEPI.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Implementation of a fast DCT-based vertical interpolation
% - Adapted and simplified for interpolating EPI's

function img_out = fdctEPI(img_in,factor)
% img_in: input image
```

```
% factor: interpolation magnitud
% It up-samples vertically img_in using fastDCT interpolation

    % Images can be any size and factor >1
    % Factors that will cause NaN:
    % 1, 1.4, 4.8, 2.2, 2.6, 3, 3.4, 3.8, 4.2, 4.6, 5, 5.4, 5.8, 6.2 and
    % rest of odd numbers
    img_in= im2double(img_in);
    cols = size(img_in,2);
    rows_in = size(img_in,1);
    rows_out = floor(rows_in*factor);

    % img_out depends on source rows and cols
    % DCT matrix depends only on source and output rows

    %Pre-allocating matrices
    components= size(img_in,3); %In case RGB image
     img_out = zeros(rows_out,cols,components);
    DCT = zeros(rows_out,rows_in);

    No = rows_in;
    n = repmat(0:rows_in-1,rows_out,1);
    k = repmat((0:rows_out-1)',1,rows_in);

    DCT = sincd(2*No-2, 2*rows_in, (k+1/2)/factor -n -1/2) .* ...
        cos( pi/(2*rows_in) * ( (k+1/2)/factor -n -1/2) ) + ...
        sincd((2*No-2), 2*rows_in, (k+1/2)/factor +n +1/2) .* ...
        cos( pi/(2*rows_in) * ( (k+1/2)/factor +n +1/2) );

    % Multiplication by DCT-Matrix to all image component layers
    for i=1:components
        img_out(:,:,i) = DCT*img_in(:,:,i);
    end

    % Truncating negative and greater than 1 values
     img_out(img_out<0) = 0;
     img_out(img_out>1) = 1;

end
```

Fichero sincd.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
```

```
% David Valle (230113) david.valle@tut.fi
% Discrete sinc function

function result = sincd(M,N,x)
% sincd(M,N,x) = sin(piMx/N)/(Nsin(pix/N))
% Digital sinc function used in fastDCT

    % Element-by-element multiplication/divisions as parameters
    % contain matrices
    result= sin(pi*M.*x./N)./(N.*sin(pi*x./N));

end
```

7.10 Código - Reconstrucción de vistas

Fichero reconstruct_views.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Recontruction of views from different EPI

function reconstruct_views(object,density,factor,v_res,interpolation)
% Object: Name of the scene
% Density: Number of views -> Vertical resolution of EPIs
% Factor: Factor of interpolation applied vertically
% v_res: Vertical resolution of views -> Number of EPIs
% interpolation: Method/kernel used for re-sampling
% It reconstructs all the views from the re-sampled EPIs

    % Concatenation of input/output path
    name_in = strcat(object,'/epis_',int2str(density),'_',interpolation,'/');
    name_out =
          strcat(object,'/parallel_',int2str(density),'_',interpolation,'/');

    for i=1:density
        for j=1:v_res
            epi = imread(strcat(name_in,int2str(j),'_',int2str(factor)),'png');
            view(j,:,:) = epi(i,:,:);
        end
        imwrite(view,strcat(name_out,int2str(i),'_',int2str(factor),'.png'));
    end
end
```

7.11 Código - Algoritmos Fast DCT

Fichero fast_dct1D.m con sus 3 versiones mejoradas.

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Implementation of a fast DCT-based interpolation

% a = SUM(n=0,N-1)an( A + B )
% A = sincd(2No-2, 2N, (k+1/2)/d -n -1/2 ) * cos(pi/2N((k+1/2)/d -n -1/2))
% B = sincd(2No-2, 2N, (k+1/2)/d +n +1/2 ) * cos(pi/2N((k+1/2)/d +n +1/2))
% No = min(N,|dN|)
% sincd(M,N,x) = sin(piMx/N)/(Nsin(pix/N))

% Explanation of variables
% a = Output Signal
% an = Input Signal
% N = signal length
% k = current output signal index
% d = scaling factor

function output = fast_dct1D(sample,factor)

    if( round( log2(size(sample,2)) ) ~= log2(size(sample,2)) )
        error('Lenght of input signal should be power of 2');
    end
    if( round( log2(factor) ) ~= log2(factor) )
        error('Factor should produce an output signal power of 2');
    end

    %1D Sample
    size_in= size(sample,2);
    size_out= size_in*factor;
    output= zeros(1,size_out);

    if factor >= 1
        No= min(size_in,ceil(factor*size_in));
    else
        No= min(size_in,floor(factor*size_in));
    end

    %Version 1
```

```
    for k= 0:size_out-1
        for n= 0:size_in-1
            part1= sincd(2*No-2, 2*size_in, (k+1/2)/factor -n -1/2);
            part3= sincd(2*No-2, 2*size_in, (k+1/2)/factor +n +1/2);
            part2= cos( (pi/(2*size_in) ) * ( (k+1/2)/factor -n -1/2) );
            part4= cos( (pi/(2*size_in) ) * ( (k+1/2)/factor +n +1/2) );
            output(k+1)= output(k+1) + sample(n+1)*(part1*part2+part3*part4);
        end
    end

    %Version 2
    DCT= zeros(size_out,size_in);
    %Calc of DCT matrix
    for k= 0:size_out-1
        DCT(k+1,1:size_in)= ...
            sincd(2*No-2, 2*size_in, (k+1/2)/factor -(0:size_in-1) -1/2) .* ...
            cos( pi/(2*size_in) * ( (k+1/2)/factor -(0:size_in-1) -1/2) ) + ...
            sincd(2*No-2, 2*size_in, (k+1/2)/factor +(0:size_in-1) +1/2) .* ...
            cos( pi/(2*size_in) * ( (k+1/2)/factor +(0:size_in-1) +1/2) );
    end
    output= (DCT*sample')';

    %Final version
    %Constructing DCT matrix
    n = repmat(0:size_in-1,size_out,1);
    k = repmat((0:size_out-1)',1,size_in);

    DCT=sincd(2*No-2, 2*size_in, (k+1/2)/factor -n -1/2) .* ...
    cos( pi/(2*size_in) * ( (k+1/2)/factor -n -1/2) ) + ...
    sincd((2*No-2), 2*size_in, (k+1/2)/factor +n +1/2) .* ...
    cos( pi/(2*size_in) * ( (k+1/2)/factor +n +1/2) );

    output= (DCT*sample')';
end
```

Fichero fast_dct2D.m con sus 3 versiones mejoradas.

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Implementation of a fast DCT-based interpolation in 2 dimensions


%function output = fast_dct2D(image,factor_x,factor_y)
```

```
function e = fast_dct2D
    t = cputime;

    %Important to convert to double, imread returns uint8.
    img= im2double(imread('chrome.png'));
    img_in= im2double(imread('chrome.png'));

    rows= size(img,1);
    cols= size(img,2);
    factor= 1;
    cols_out= cols*factor;
    img_rows= zeros(rows,cols_out);
    rows_out= rows*factor;
    img_out= zeros(rows_out,cols_out);

    % Version 1
    for n=1:1
        %Zoom-in
         for i=1:rows
             img_rows(i,:)= fast_dct1D(img_in(i,:),factor);
        end
        for i=1:cols_out
            img_out(:,i)= fast_dct1D(img_rows(:,i).',factor).';
        end

        %Zoom-out
        for i=1:cols_out
            img_rows(:,i)= fast_dct1D(img_out(:,i).',1/factor).';
        end
         for i=1:rows
             img_in(i,:)= fast_dct1D(img_rows(i,:),1/factor);
        end
    end

    % Version 2
    %Pre-allocating matrices
     img_out= zeros(size_out,size_out);
    DCT= zeros(size_out,size_in);
    for k= 0:size_out-1
        DCT(k+1,1:size_in)= ...
            sincd(2*No-2, 2*size_in, (k+1/2)/factor -(0:size_in-1) -1/2 ) .* ...
            cos( pi/(2*size_in) * ( (k+1/2)/factor -(0:size_in-1) -1/2 ) ) + ...
            sincd(2*No-2, 2*size_in, (k+1/2)/factor +(0:size_in-1) +1/2 ) .* ...
            cos( pi/(2*size_in) * ( (k+1/2)/factor +(0:size_in-1) +1/2 ) );
```

```
    end

     % Final version
    n = repmat(0:size_in-1,size_out,1);
    k = repmat((0:size_out-1)',1,size_in);
    DCT=sincd(2*No-2, 2*size_in, (k+1/2)/factor -n -1/2) .* ...
    cos( pi/(2*size_in) * ( (k+1/2)/factor -n -1/2) ) + ...
    sincd((2*No-2), 2*size_in, (k+1/2)/factor +n +1/2) .* ...
    cos( pi/(2*size_in) * ( (k+1/2)/factor +n +1/2) );


    img_out= DCT*img_in*DCT';

    % Truncating negative and greater than 1 values
     img_out(img_out<0)= 0;
     img_out(img_out>1)= 1;
end
```

7.12 Código - Tests Fast DCT

Fichero run_fdct1D.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Testing Fast DCT-based interpolation

%Ramp signal
signal= 1:128;
factor= 3;
figure
output= fastdct1D(signal,factor);
subplot(1,2,1), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor %i',factor));
output= fdct1D(signal,1/factor);
subplot(1,2,2), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor 1/%i',factor));
legend('input','output');
clear all

%Ramp with step signal
signal= 1:128;
for i= 64:96
    signal(i)= i-20;
end
factor= 2;
figure
output= fastdct1D(signal,factor);
subplot(1,2,1), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor %i',factor));
output= fdct1D(signal,1/factor);
subplot(1,2,2), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor 1/%i',factor));
legend('input','output');
clear all

%Sinusoidal signal
signal= sind(1:512);
factor= 2;
figure
output= fastdct1D(signal,factor);
```

```
subplot(1,2,1), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor %i',factor));
output= fdct1D(signal,1/factor);
subplot(1,2,2), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor 1/%i',factor));
legend('input','output');
clear all

%Pulse signal
for i=1:64
    signal(i)= 20;
end
for i=65:192
    signal(i)= 200;
end
for i=193:256
    signal(i)= 20;
end
factor= 2;
figure
output= fastdct1D(signal,factor);
subplot(1,2,1), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor %i',factor));
output= fdct1D(signal,1/factor);
subplot(1,2,2), plot(1:size(signal,2),signal,'b',1:size(output,2),output,'r');
title(sprintf('Fast-DCT interpolation factor 1/%i',factor));
legend('input','output');
clear all
```

Fichero run_fdct2D.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Testing Fast DCT-based interpolation

clear all;
tic;
img_in=im2double(imread('text.png'));
out2=img_in;
for i=1:1000
    out=fdct2D(out2,2);
    out2=fdct2D(out,2);
end
```

```
toc;
figure; imshow(img_in);
figure; imshow(out);
imwrite(out,'text75_out.png');
figure; imshow(out2);
imwrite(out2,'text75_out2.png');
```

Fichero population_test.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Testing fastest way to populate matrix with given parameters
% Parameters are according renders we should take
% Around 600x600pixels and factor [2:10]

clear all
times= [];
tests= [2:10; repmat([300; 600; 900],1,9)];

 for j=1:size(tests,2)
     for i=2:size(tests,1)
        factor= tests(1,j);
        size_in= tests(i,j);
        size_out=factor*size_in;

        clear ns ks
        tic;
        ns = kron([1:size_in]-1,ones(1,size_out)');
        ks = kron(ones(1,size_in),[1:size_out]'-1);
        times(j,i-1) = toc;

        clear ns ks
        tic;
        ns = repmat([0:size_in-1],size_out,1);
        ks = repmat([0:size_out-1]',1,size_in);
        times(j,i+2) = toc;

        clear ns ks
        tic;
        temp= [0:size_in-1];
        ns= temp(ones(1,size_out),:);
        temp= [0:size_out-1]';
        ks= temp(:,ones(1,size_in));
```

```
        times(j,i+5) = toc;
     end
 end

figure;
bar(times,'grouped');
title('Comparison population of matrix','FontWeight','bold');
times_leg =  legend('kron300','kron600','kron900', ...
        'repmat300','repmat600','repmat900', ...
        'ones300','ones600','ones900');
set(times_leg,'Location','NorthWest')
xlabel('Cases: 2 to 10 factor, 300 - 600 - 900pixels');
ylabel('Seconds');
```

7.13 Código - Tests

Fichero run_epi_interp.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi

function run_epi_interp(object)
% Object: Name of the scene
% It runs epi_interp() with 5 factors [2, 4, 8, 16, 32] and
% 5 interpolation kernels [triangle, cubic, lanczos2, lanczos3, fastdct]

disp('Processing epi_interp factor 2 density 256');
epi_interp(object,512,2,600,'triangle');
epi_interp(object,512,2,600,'cubic');
epi_interp(object,512,2,600,'lanczos2');
epi_interp(object,512,2,600,'lanczos3');
epi_interp(object,512,2,600,'fastdct');

disp('Processing epi_interp factor 4 density 128');
epi_interp(object,512,4,600,'triangle');
epi_interp(object,512,4,600,'cubic');
epi_interp(object,512,4,600,'lanczos2');
epi_interp(object,512,4,600,'lanczos3');
epi_interp(object,512,4,600,'fastdct');

disp('Processing epi_interp factor 8 density 64');
epi_interp(object,512,8,600,'triangle');
epi_interp(object,512,8,600,'cubic');
epi_interp(object,512,8,600,'lanczos2');
epi_interp(object,512,8,600,'lanczos3');
epi_interp(object,512,8,600,'fastdct');

disp('Processing epi_interp factor 16 density 32');
epi_interp(object,512,16,600,'triangle');
epi_interp(object,512,16,600,'cubic');
epi_interp(object,512,16,600,'lanczos2');
epi_interp(object,512,16,600,'lanczos3');
epi_interp(object,512,16,600,'fastdct');

disp('Processing epi_interp factor 32 density 16');
epi_interp(object,512,32,600,'triangle');
```

```
epi_interp(object,512,32,600,'cubic');
epi_interp(object,512,32,600,'lanczos2');
epi_interp(object,512,32,600,'lanczos3');
epi_interp(object,512,32,600,'fastdct');

figure;
timing= reshape(timing,5,6)';
plot(timing);
title('Interpolation time comparison','FontWeight','bold');
timing_leg =  legend('triangle','cubic','lanczos2', 'lanczos3','fastdct');
set(timing_leg,'Location','NorthWest')
xlabel('Interpolation factors: 32, 16, 8, 6, 4, 2');
ylabel('Seconds');

end
```

Fichero run_reconstruct_views.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Script that runs reconstruct_view with all possible configurations

function run_reconstruct_views(object)
% object: Name of the scene
% It runs reconstruct_views() with 5 factors [2, 4, 8, 16, 32] and
% 5 interpolation kernels [triangle, cubic, lanczos2, lanczos3, fastdct]

disp('Processing reconstruct_views factor 2 density 256');
reconstruct_views(object,512,2,600,'triangle');
reconstruct_views(object,512,2,600,'cubic');
reconstruct_views(object,512,2,600,'lanczos2');
reconstruct_views(object,512,2,600,'lanczos3');
reconstruct_views(object,512,2,600,'fastdct');

disp('Processing reconstruct_views factor 4 density 128');
reconstruct_views(object,512,4,600,'triangle');
reconstruct_views(object,512,4,600,'cubic');
reconstruct_views(object,512,4,600,'lanczos2');
reconstruct_views(object,512,4,600,'lanczos3');
reconstruct_views(object,512,4,600,'fastdct');

disp('Processing reconstruct_views factor 8 density 64');
reconstruct_views(object,512,8,600,'triangle');
```

```
reconstruct_views(object,512,8,600,'cubic');
reconstruct_views(object,512,8,600,'lanczos2');
reconstruct_views(object,512,8,600,'lanczos3');
reconstruct_views(object,512,8,600,'fastdct');

disp('Processing reconstruct_views factor 16 density 32');
reconstruct_views(object,512,16,600,'triangle');
reconstruct_views(object,512,16,600,'cubic');
reconstruct_views(object,512,16,600,'lanczos2');
reconstruct_views(object,512,16,600,'lanczos3');
reconstruct_views(object,512,16,600,'fastdct');

disp('Processing reconstruct_views factor 32 density 16');
reconstruct_views(object,512,32,600,'triangle');
reconstruct_views(object,512,32,600,'cubic');
reconstruct_views(object,512,32,600,'lanczos2');
reconstruct_views(object,512,32,600,'lanczos3');
reconstruct_views(object,512,32,600,'fastdct');

figure;
timing= reshape(timing,5,6)';
plot(timing);
title('Interpolation time comparison','FontWeight','bold');
timing_leg =  legend('triangle','cubic','lanczos2', 'lanczos3','fastdct');
set(timing_leg,'Location','NorthWest')
xlabel('Interpolation factors: 32, 16, 8, 6, 4, 2');
ylabel('Seconds');

end
```

Fichero test_psnr.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Script that runs epi with all possible configurations
% It saves process times in file "times.mat"

function test_psnr(object,density,factor)
% Object: Name of the scene
% Density: Number of views
% Factor: Interpolation magnitud
% It calculates the PSNR for every factor and view
```

```
PSNR = zeros(5,512);

for i=1:density
    i
    original= double(imread(strcat(object,'/parallel_rc_',
                           int2str(density),'/',int2str(i)),'png'));

    image1 =
       double(imread(strcat(object,'/parallel_',int2str(density),'_triangle/'
                                ,int2str(i),'_',int2str(factor)),'png'));
    image2 =
       double(imread(strcat(object,'/parallel_',int2str(density),'_cubic/',
                                int2str(i),'_',int2str(factor)),'png'));
    image3 =
       double(imread(strcat(object,'/parallel_',int2str(density),'_lanczos2/',
                                int2str(i),'_',int2str(factor)),'png'));
    image4 =
       double(imread(strcat(object,'/parallel_',int2str(density),'_lanczos3/',
                                int2str(i),'_',int2str(factor)),'png'));
    image5 =
       double(imread(strcat(object,'/parallel_',int2str(density),'_fastdct/',
                                int2str(i),'_',int2str(factor)),'png'));

    PSNR(i,1) = 10 * log10(255^2/mean2((original-image1).^2));
    PSNR(i,2) = 10 * log10(255^2/mean2((original-image2).^2));
    PSNR(i,3) = 10 * log10(255^2/mean2((original-image3).^2));
    PSNR(i,4) = 10 * log10(255^2/mean2((original-image4).^2));
    PSNR(i,5) = 10 * log10(255^2/mean2((original-image5).^2));

end

figure;
hold on;
plot(PSNR);
title('PSNR comparison','FontWeight','bold');
timing_leg =  legend('triangle','cubic','lanczos2', 'lanczos3','fastdct');
set(timing_leg,'Location','SouthEast')
xlabel('Different views');
ylabel('Squared error loss');

end
```

Fichero test_psnr_average.m

```
% Tampere University of Technology (TUT)
% Departament of Signal Processing (SGN)
% David Valle (230113) david.valle@tut.fi
% Script that runs epi with all possible configurations
% It saves process times in file "times.mat"

function PSNR = test_psnr_average(object,density)
% Object: Name of the scene
% Density: Number of views
% Factor: Interpolation magnitud
% It calculates the PSNR for every factor and
% its average between all the views

PSNR = zeros(5,5);
factors = [2, 4, 8, 16, 32];

for i=1:size(factors,2)
    i
    for j=1:density
        j
        original= double(imread(strcat(object,'/parallel_rc_',int2str(density),
                                       '/',int2str(j)),'png'));

        image1 =
           double(imread(strcat(object,'/parallel_',int2str(density),
                        '_triangle/',int2str(j),'_',int2str(factors(i))),'png'));
        image2 =
           double(imread(strcat(object,'/parallel_',int2str(density),
                        '_cubic/',int2str(j),'_',int2str(factors(i))),'png'));
        image3 =
           double(imread(strcat(object,'/parallel_',int2str(density),
                        '_lanczos2/',int2str(j),'_',int2str(factors(i))),'png'));
        image4 =
           double(imread(strcat(object,'/parallel_',int2str(density),
                        '_lanczos3/',int2str(j),'_',int2str(factors(i))),'png'));
        image5 =
           double(imread(strcat(object,'/parallel_',int2str(density),
                        '_fastdct/',int2str(j),'_',int2str(factors(i))),'png'));

        PSNR(i,1) = PSNR(i,1) + 10 * log10(255^2/mean2((original-image1).^2))
        PSNR(i,2) = PSNR(i,2) + 10 * log10(255^2/mean2((original-image2).^2));
        PSNR(i,3) = PSNR(i,3) + 10 * log10(255^2/mean2((original-image3).^2));
        PSNR(i,4) = PSNR(i,4) + 10 * log10(255^2/mean2((original-image4).^2));
        PSNR(i,5) = PSNR(i,5) + 10 * log10(255^2/mean2((original-image5).^2));
```

```
    end
end

PSNR = PSNR / density;

figure;
hold on;
plot(PSNR,'.');
title('PSNR comparisong of every factor','FontWeight','bold');
timing_leg =  legend('triangle','cubic','lanczos2', 'lanczos3','fastdct');
set(timing_leg,'Location','NorthEast')
xlabel('Interpolation factors: 2, 4, 8, 16, 32');
ylabel('Squared error loss');

end
```

7.14 Gráficas de los tests

Las siguientes gráficas muestran los resultados PSNR de cada vista para cada método de interpolación. Cada gráfica representa un factor de interpolación diferente.

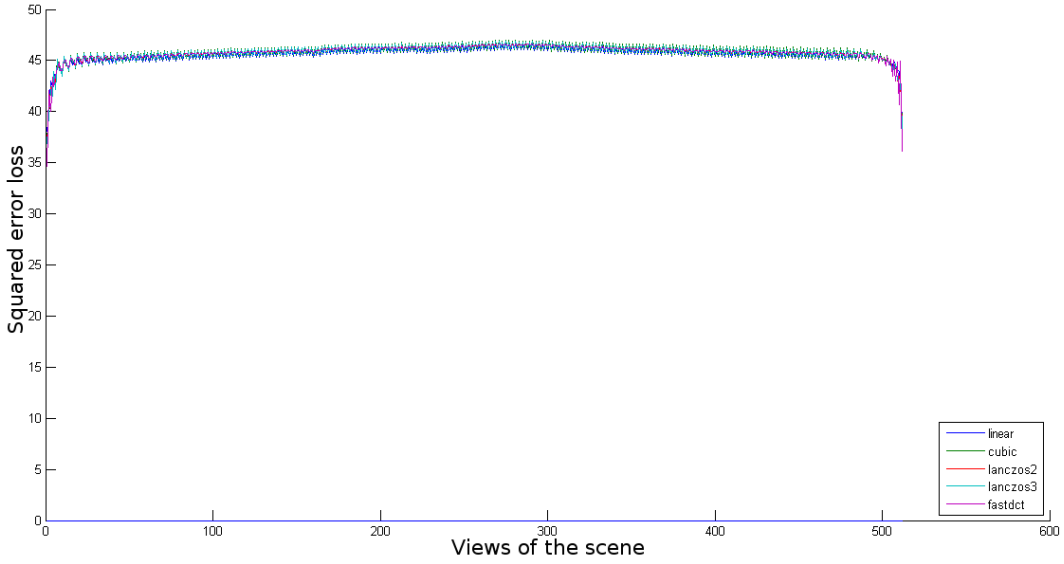


Figura 7.2: PSNR de las vistas reconstruidas con cada método de interpolacón, factor 2.

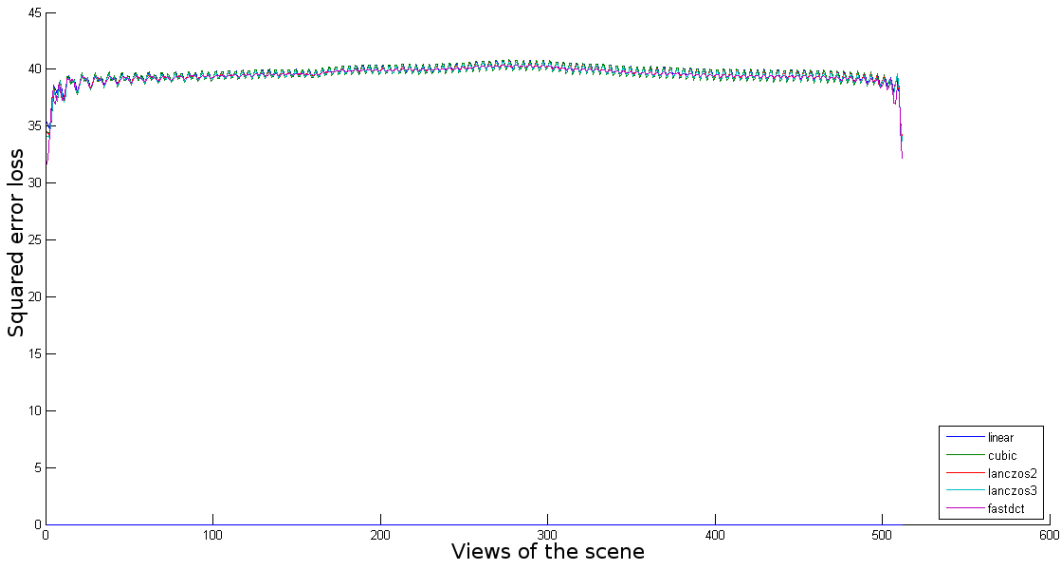


Figura 7.3: PSNR de las vistas reconstruidas con cada método de interpolación, factor 4.

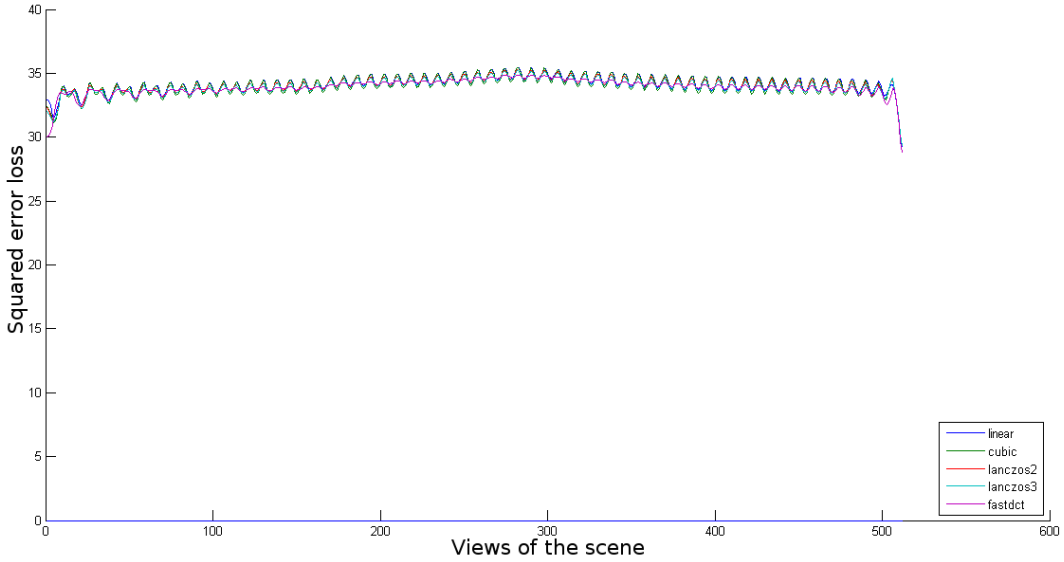


Figura 7.4: PSNR de las vistas reconstruidas con cada método de interpolación, factor 8.

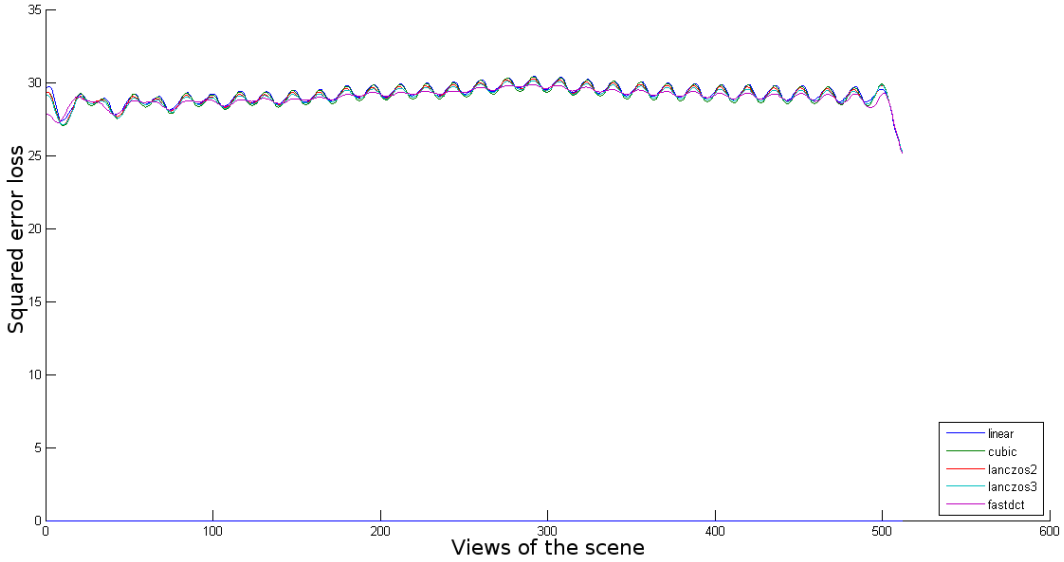


Figura 7.5: PSNR de las vistas reconstruidas con cada método de interpolación, factor 16.

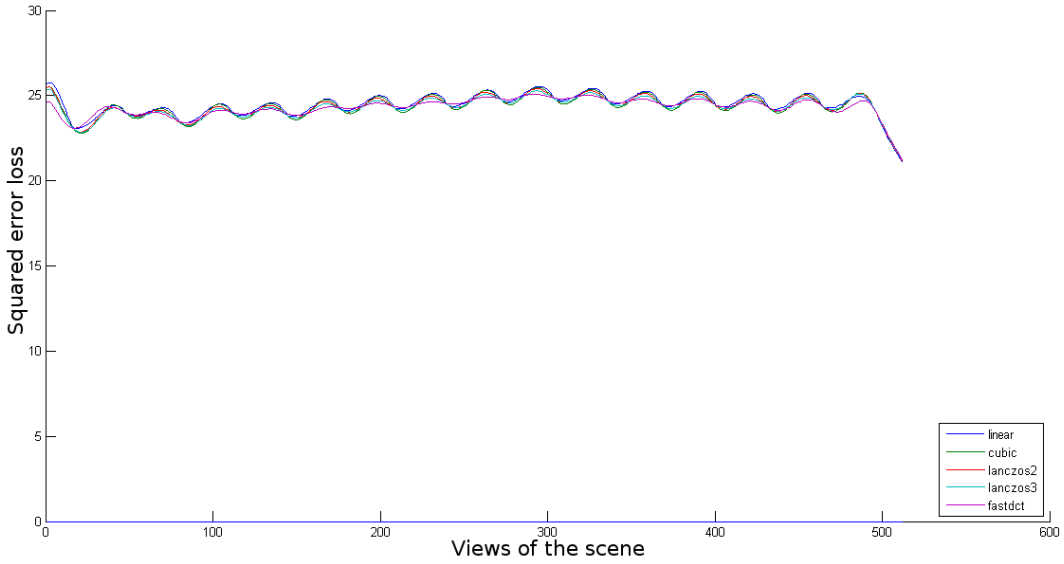


Figura 7.6: PSNR de las vistas reconstruidas con cada método de interpolación, factor 32.