Università degli Studi di Salerno

Compressione Dati

Hologram Compression Lossless

Professore

Bruno Carpentieri

Studenti

Cetrangolo Mario

Santonastaso Manlio

Anno Accademico 2021/2022

Sommario

Introduzione

Gli ologrammi sono definiti come figure o pattern d'onda interferenti ottenute tramite l'uso di un laser, aventi la specificità di creare un effetto fotografico tridimensionale: essi, a differenza delle normali fotografie, ci mostrano una rappresentazione tridimensionale dell'oggetto proiettato.

Qualche mese fa *Meta* di Mark Zuckerberg ha lanciato la realtà virtuale consentirà di "tele trasportarci al lavoro, a un concerto o ad una riunione di famiglia in forma di ologramma", facendo risparmiare tempo, traffico e non incidendo sull'ambiente.

Di recente si è venuto a conoscenza che Google introdurrà le videochiamate olografiche nel *Project Starline*, il quale garantisce videochiamate in altissima definizione con immagini 3D che non si deformano a seconda dell'angolazione, rendendo realtà una tecnologia che abbiamo visto in tanti titoli di fantascienza.

Diverse e importanti le applicazioni anche fuori dallo *showbiz*. In campo militare, le mappe olografiche vengono già utilizzate per la ricognizione e lo studio dettagliato degli scenari di ingaggio. Anche il matrimonio tra medicina diagnostica e olografia sta dando buoni risultati. Partendo da una tac, si può oggi ricostruire un'immagine tridimensionale dell'organo interessato, con risvolti positivi tanto dal punto di vista analitico quanto da quello della ridotta invasività.

Poiché le statistiche e le proprietà dei segnali olografici differiscono notevolmente dalle immagini naturali come le fotografie, le soluzioni di codifica convenzionali non sono ottime. Ad oggi si hanno problemi nel salvataggio e trasmissione degli ologrammi dovuti alla loro dimensione.

Pertanto, sono necessarie nuove soluzioni di codifica e trasformazioni per comprimere gli ologrammi in modo più efficace. Negli ultimi anni sono state proposte diverse tecniche per affrontare questo problema: ***Fresnelets*** [1], ***wavelet-bandelets*** [2], ***directional-adaptive wavelets and arbitrary packet decompositions*** [3], ***vector quantisation lifting schemes*** [4], ***wave atom transforms*** [5], ***mode dependent directional transform-***

based HEVC [6], ***overcomplete Gabor wavelet dictionary*** [7] ***and nonlinear canonical transforms*** [8]

In particolare, utilizzeremo il principio ***object plane coding*** [9], [10]: propagando all'indietro il campo d'onda dell'ologramma usando la trasformata di Fresnel al piano dell'oggetto, rifocalizziamo efficacemente l'ologramma su una rappresentazione simile a un'immagine, rendendola successivamente meglio comprimibile mediante trasformazioni e codec convenzionali utilizzando precisamente algoritmo di compressione JPEG 2000 e JPEG-LS in modo tale da effettuare un confronto tra i due algoritmi.

Capitolo 1

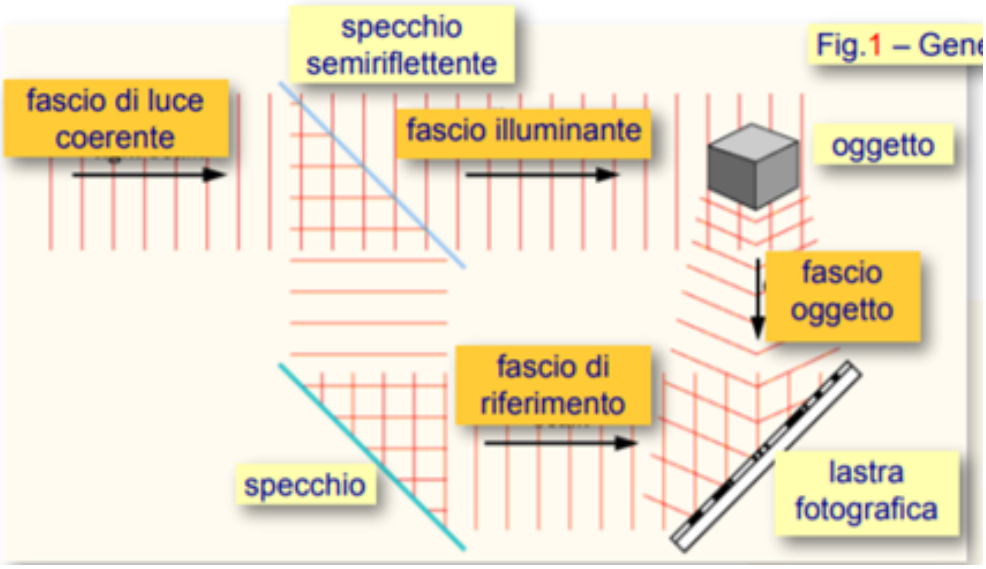
1. Olografia

Il termine ologramma indica una **specifica tecnica di visualizzazione delle immagini in tre dimensioni**, o, in senso tecnico, la creazione di una immagine *parallattica* tramite l'utilizzo di fasci di luce laser. Più in generale, gli ologrammi sono immagini tridimensionali stereoscopiche, che assumono prospettive diverse a seconda del punto in cui le si osserva.

I primi ologrammi furono inizialmente teorizzati e poi inventati dal fisico ungherese *Dennis Gabor*, tramite l'utilizzo di fasci di luce verde generati da una lampada di vapori di mercurio.

1.1 Generazione dell'ologramma

La *caratteristica costanza della luce laser* è quello che, infatti, permette di riprodurre con precisione un'immagine precedentemente registrata, mediante la combinazione di due fasci laser: il primo viene diffratto dall'oggetto, mentre il secondo viene riflesso da uno specchio; i due fasci laser, vengono poi fatti interferire tra di loro. È così che si genera un effetto fotografico tridimensionale che permette di percepire anche la terza dimensione, a differenza della semplice proiezione su una superficie di un'immagine. [11]



Ecco nel dettaglio come l'ologramma viene creato tramite un vero e proprio *gioco di specchi:*

- nella fase di registrazione dell'immagine da riprodurre, si invia un fascio di luce laser verso l'oggetto da riprodurre;
- il fascio di luce viene inviato anche verso una lastra di materiale sensibile;
- la luce laser interferisce con quella riflessa dall'oggetto. Le lunghezze dei due percorsi ottici (quello che illumina direttamente alla lastra e quello che viene ad essa riflesso dall'oggetto) dovranno essere quanto più possibile simili tra loro, o la loro differenza essere molto inferiore alla lunghezza di coerenza della sorgente utilizzata;
- sulla lastra di materiale sensibile vanno a formarsi delle linee, denominate frange di interferenza (assimilabili alle increspature che si formano su uno specchio d'acqua, o all'incontro tra più onde sonore);
- l'immagine, così registrata, conterrà tutte le informazioni sull'oggetto che si intende riprodurre all'interno del pattern di interferenza, che registra l'intensità e la fase della luce proveniente dall'oggetto;
- facendo passare attraverso le fenditure delle frange di interferenza un fascio di luce laser, quest'ultimo sarà diffratto, generando delle onde luminose che si andranno a sovrapporre l'una sull'altra, ricostruendo l'immagine tridimensionale dell'oggetto originariamente registrato.

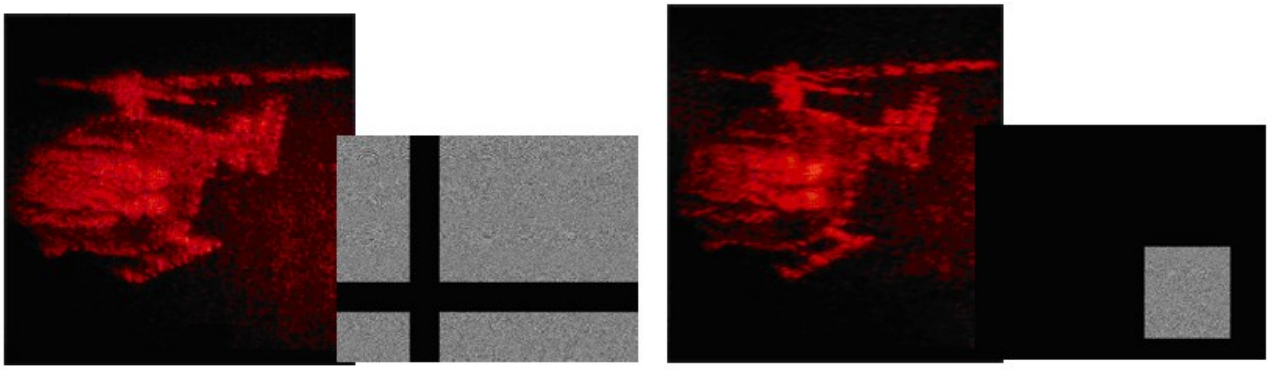
Tramite tale processo, lo spettatore potrà, così, vedere in tre dimensioni l'immagine come se fosse fisicamente presente.

1.2 Rappresentazione dell'ologramma

Si potrebbe pensare che il ricorso a matrici di dati bidimensionali, cioè all'ologramma, per rappresentare immagini tridimensionali, possa essere uno strumento facilmente utilizzabile per la loro trasmissione e memorizzazione.

In realtà, la riproduzione di alta qualità dell'immagine oggetto richiede un'altissima definizione per l'ologramma e conseguentemente una quantità elevata di informazione (dell'ordine di grandezza di 1 TB per ologrammi ad alta risoluzione); peraltro è possibile ridurre tale quantità di informazione accettando una diminuzione della qualità dell'immagine riprodotta. Uno dei metodi possibili consiste nel considerare solo una parte dell'ologramma originale a sotto campionamento. Un altro metodo, che si presta anche alla realizzazione di sistemi scalabili, consiste nel considerare solo una parte dei campioni di un ologramma; la qualità ottenibile per l'immagine riprodotta dipende solo dalla quantità di campioni considerata. [11] Come si può notare in questa serie di immagini.

Fig. 19 – Riproduzione ottenuta con parte di un ologramma [11].



1.3 Difficoltà di codifica

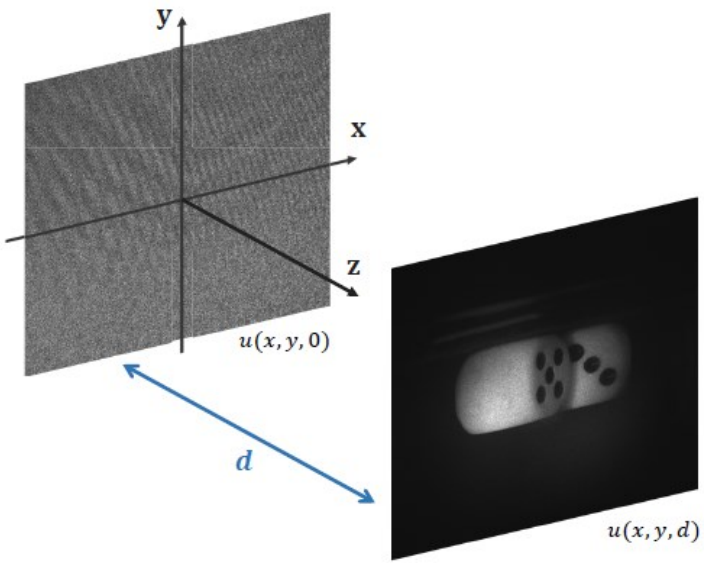
Le caratteristiche statistiche dei campioni ricavati da un ologramma sono differenti da quella dei campioni ricavati dalle immagini bidimensionali convenzionali; pertanto, non è possibile applicare direttamente gli stessi algoritmi di compressione per la diminuzione dell'informazione associata. Gli ologrammi contengono dell'informazione non rilevante ai fini della produzione dell'immagine desiderata, che può essere rimossa con un opportuno filtraggio *band-pass* nel campo delle frequenze. L'applicazione di trasformate ai dati così filtrati risulta una codifica efficace. Ad esempio, gli ologrammi hanno un contenuto molto più ad alta frequenza, spesso possiedono ***rumore speckle*** (si ottiene quando un'onda coerente viene fatta passare attraverso un mezzo disordinato) e le informazioni localizzate nella scena 3D si diffrangeranno, si diffonderanno e potenzialmente influenzeranno tutti i pixel dell'ologramma digitale. Ecco perché gli algoritmi di rappresentazione e codifica convenzionali, come le famiglie di codec standard JPEG e MPEG, non sono ottimali per la codifica di ologrammi. Inoltre, gli ologrammi per display olografici multiutente richiedono risoluzioni di diversi Gigapixel, aumentando la necessità di metodologie di compressioni efficienti.

Capitolo 2

2. Lavoro correlato

Per la realizzazione di questo progetto è stato preso in considerazione il lavoro di due professori dell'università di Brussel (Belgio) *David Blinder e Peter Schelkens* intitolato ***"Integer Fresnel Transform for Lossless Hologram Compression"***. [12]

In questo articolo è stata proposta la trasformata intera di ***Fresnel***, che è stata la prima trasformata senza perdita di dati progettata per la codifica degli ologrammi. Nel prossimo capitolo spiegheremo come implementare la variante intera senza perdita di dati della trasformata di Fresnel.



Gli autori hanno utilizzato due tecniche per calcolare la trasformata di *Fresnel*, ***Convolutional Form*** ed il ***Fourier Form***. La principale differenza tra le tecniche è la relazione tra il *pixel pitch* del piano sorgente e la *destination plane*.

- Il *pixel pitch* è la distanza in millimetri dal centro di un pixel al centro del pixel adiacente.
- *Destination plane* è la distanza fisica di campionamento tra i successivi centri di pixel dell'ologramma regolarmente campionato.

A seconda delle dimensioni (virtuali) dell'oggetto, del posizionamento dell'ologramma e dalla geometria della scena, un metodo sarà spesso preferibile per visualizzare meglio l'oggetto ricostruito.

La forma convolutiva FD conserva il *pixel pitch*, mentre forma di *Fourier* cambierà *il pixel pitch* del piano di destinazione a seconda della lunghezza d'onda e della distanza di propagazione.

Queste due forme sono trasformate numeriche e non possono essere utilizzate per la codifica lossless, perché non sono definite per implementazioni intere.

Il loro obiettivo è stato quello di sviluppare una variante intera della trasformata di Fresnel per la codifica lossless di ologrammi digitali. Integrando la trasformata di Fresnel con JPEG 2000, ottenendo un risparmio di bit rate compresso tra 0,12 e 2,83 bpc rispetto all'implementazione predefinita di JPEG 2000 lossless.

Capitolo 3

3. Argomenti trattati

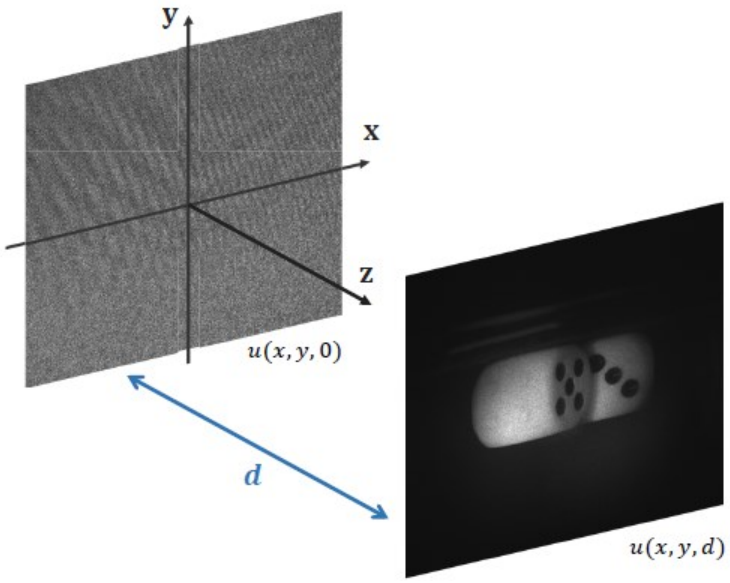
3.1 Trasformata (diffrazione) di Fresnel

La diffrazione di *Fresnel* è un'approssimazione della diffrazione di *Kirchhoff-Fresnel* che può essere applicata alla propagazione delle onde del campo vicino. Viene utilizzata per calcolare il modello diffrazione creato dalle onde che passano attraverso un'apertura o intorno ad un oggetto. [13] La teoria della diffrazione scalare può modellare la propagazione della luce, descrivendo matematicamente come il campo d'onda a valori complessi u(x, y, z) evolvono nello spazio. Approssimazione di Fresnel viene usata per modellare la diffrazione tra piani paralleli.

$$u(x,y,z+d) = \frac{\exp(ikd)}{i\lambda d} \iint_{\mathbb{R}^2} u(x',y',z) \exp\left(\frac{ik}{2d}\left[(x-x')^2+(y-y')^2\right]\right) dx'dy'$$

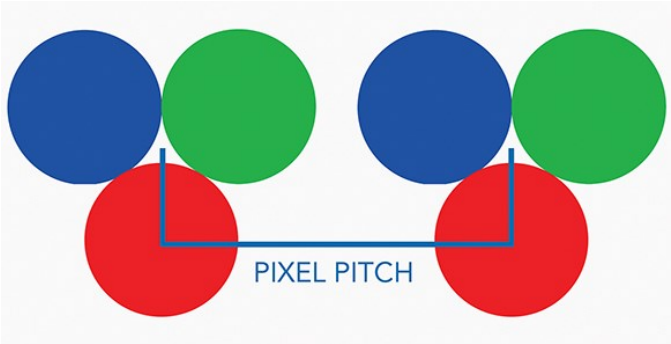
Dove:

- **u (x', y', z)** è il campo elettrico all'apertura
- **k** è il numero d'onda definito in questo modo $\boldsymbol{k = \frac{2\pi}{\Lambda}}$
- **i** è unità immaginaria



La figura sopra identifica il piano dell'ologramma è centrato nell'origine, dove u(x,y,0) è disegnato, questo rappresenta ologramma che deve essere compresso. La direzione di propagazione "z" è perpendicolare al piano dell'ologramma. Utilizzando operatore di Fresnel con distanza di propagazione interplanare D, u(x,y,d) può essere calcolato.

Esistono diverse tecniche per calcolare la trasformata di Fresnel. Nel progetto sono state utilizzate le due implementazioni numeriche più comuni, *convolutional form e Fourier form*. Le principali differenze tra le tecniche è la relazione tra *il pixel pitch* del piano sorgente (**PS**) e *destination plane* (**PD**).



Il *pixel pitch* come si può notare dalla figura è la distanza in millimetri tra i centri di pixel adiacenti. Poiché il *pixel pitch* indica la quantità di spazio tra due pixel, un *pixel pitch* più piccolo significa che c'è meno spazio vuoto tra i pixel che equivale ad una maggiore densità di pixel e una migliore risoluzione dello ologramma.

La *destination plane* è la distanza di campionamento tra i successivi centri di pixel dell'ologramma regolarmente campionato.

3.2 Convolutional Form

La forma di convoluzione della diffrazione di Fresnel è espressa da:

$$u_2(x_2,y_2) = \frac{\exp(i\frac{2\pi}{\lambda}z)}{i\lambda z}\int\int_{-\infty}^{+\infty} u_1(x_1,y_1) \exp(i\frac{\pi}{\lambda z}((x_2-x_1)^2+(y_2-y_1)^2))dx_1dy_1$$

L'equazione di cui sopra è la forma di convoluzione e può essere espressa rispetto alla seguente equazione secondo il teorema di convoluzione:

$$u_2(x_2,y_2) = \frac{\exp(i\frac{2\pi}{\lambda}z)}{i\lambda z}\mathcal{F}^{-1}\left[\mathcal{F}\left[u(x_1,y_1)\right]\mathcal{F}\left[h_F(x_1,y_1)\right]\right]$$

Dove, l'operatore $F=[\cdot]$ e $F^{-1}=[\cdot]$ indica *Fourier* e *Fourier* Inversa, rispettivamente $h_f(x,y)$ è la funzione di risposta all'impulso definita in questo modo

$$h_F(x,y) = \exp(i\frac{\pi}{\lambda z}(x^2+y^2))$$

In un'implementazione numerica, si ha bisogno valori spaziali e usiamo la ***FFT*** invece della trasformata Fourier come segue: Le variabili spaziali discrete sono definite in questo modo ***(x1,y1) = (m1Δx1,n1Δy1)*** dove **Δx1** e **Δx2** sono pixel di campionamento e m1,n1 sono indici interi del piano sorgente.

Le variabili spaziali discrete sono ***(x2,y2) = (m2Δx2,n2Δy2)***, dove **Δx2** e **Δy2** sono pixel di campionamento e **m2, n2** sono indici interi del piano di destinazione. L'intervallo degli indici interi è definito come segue

$$-\frac{N_x}{2} \le m_1, m_2 < -\frac{N_x}{2} - 1,$$
$$-\frac{N_y}{2} \le m_1, m_2 < -\frac{N_y}{2} - 1$$

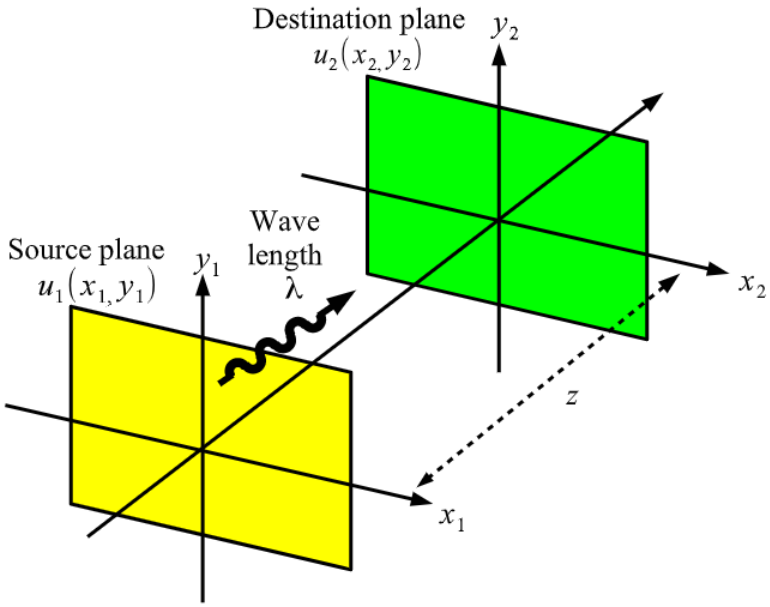
Dove, **Nx** e **Ny** sono il numero di pixel verticali e orizzontali del piano sorgente e piano di destinazione.

La diffrazione *Fresnel* discreta della forma convolutiva è come segue:

$$u_2(m_2, n_2) = \frac{\exp(i\frac{2\pi}{\lambda}z)}{i\lambda z} FFT^{-1}\Big[FFT\Big[u(m_1, m_1)\Big] FFT\Big[h_F(m_1, m_1)\Big]\Big]$$

$$h_F(m, n) = \exp(i\frac{\pi}{\lambda z}((m\Delta x_1)^2 + (n\Delta y_1)^2))$$

Notare che i pixel di campionamento del piano di destinazione sono gli stessi del piano sorgente dopo la diffrazione, ossia **Δx2 = Δx1 e Δy2 = Δy1**. [14]



3.3 Fourier Form

Si può ottenere la forma Fourier della diffrazione di *Fresnel* attraverso il ***termine quadratico***

$$u_2(x_2,y_2) = \frac{\exp(i\frac{2\pi}{\lambda}z)}{i\lambda z}\int\int_{-\infty}^{+\infty} u_1(x_1,y_1)$$
$$\exp(i\frac{\pi}{\lambda z}((x_2-x_1)^2+(y_2-y_1)^2))dx_1dy_1$$

In questa forma è espressa:

$$u(x_2,y_2) = \frac{\exp(i\frac{2\pi}{\lambda}z)}{i\lambda z}\exp(i\frac{\pi}{\lambda z}(x_2^2+y_2^2))$$
$$\int\int_{-\infty}^{+\infty} u'(x_1,y_1)\exp(-i\frac{2\pi}{\lambda z}(x_2x_1+y_2x_1))dx_1dy_1,$$

Dove **u'(x1,y1)** è definito come segue:

$$u'(x_1,y_1) = u(x_1,y_1)\exp(i\pi\frac{(x_1^2+y_1^2)}{\lambda z}).$$

Questa forma è stata riscritta dalla Trasformata di Fourier. La diffrazione di *Fresnel* Discreta in forma Fourier è espressa:

$$u(m_2,n_2) = \frac{\exp(i\frac{2\pi}{\lambda}z)}{i\lambda z}\exp(i\frac{\pi}{\lambda z}((m_2\Delta x_2)^2+(n_2\Delta y_2)^2))$$
$$FFT\left[u'(m_1,n_1)\right]$$

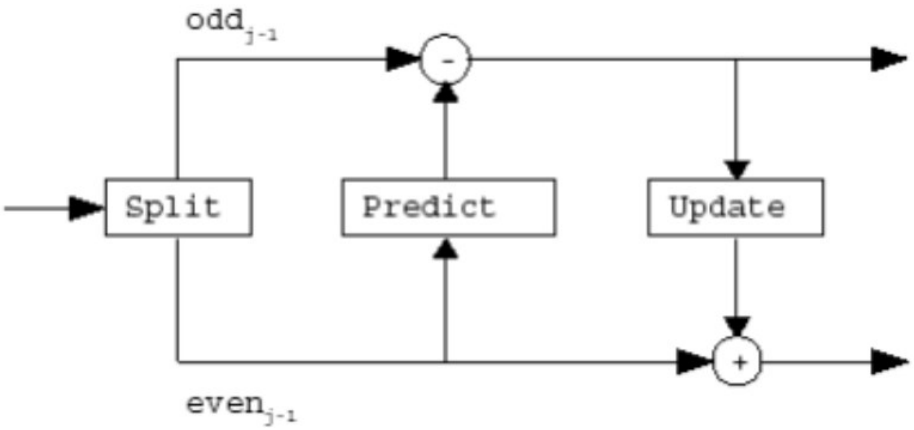
Notare che lo spazio di campionamento del piano di destinazione è ridimensionato a **Δx2 = Δx1/ (λz)** e **Δy2 = Δy1/(λz)**. [14]

3.4 Lifting Scheme

Nella forma generale della trasformata di *Fresnel* (essendo unaria, grazie all'utilizzo del ***lifting scheme*** si possono applicare le trasformate viste in precedenza), avendo in ingresso **X(matrice)** partizionato in due parti $\mathbf{a_0}$ e $\mathbf{b_0}$. Su di esse effettua una cascata di funzioni reversibili **Pj** (**prediction**) e **Uj** (**update**) applicate alternativamente su tutti gli elementi di $\mathbf{a_j}$ e $\mathbf{b_j}$ questo modo:

$$b_{j+1} = \mathrm{P}_j(a_j, b_j) \quad \text{and} \quad a_{j+1} = \mathrm{U}_j(a_j, b_{j+1})$$

Precisamente la funzione di *Prediction* calcola una previsione per i campioni dispari, in base ai campioni pari (o viceversa). Questa previsione viene sottratta dai campioni dispari, creando un segnale di errore, mentre la funzione di Update viene utilizzata per "preparare" il segnale per la fase di previsione successiva. Utilizza i campioni dispari previsti per preparare quelli pari. Questo aggiornamento viene sottratto dai campioni pari, producendo un segnale.



Dove $\mathbf{P_j(a_j , \cdot)}$ e **Uj (· , $\mathbf{b_j + 1}$)** devono essere biiezioni per qualsiasi segnale intero **aj** o **bj+1**, ∀**j** ∈ **{0, 1, …,m}**.

In genere, la seguente implementazione viene utilizzata per la codifica *lossless*:

$$\mathrm{P}_j(a_j, b_j) = [\Pi_j(a_j)] \pm b_j \quad \text{and} \quad \mathrm{U}_j(a_j, b_{j+1}) = [\Upsilon_j(b_{j+1})] \pm a_j$$

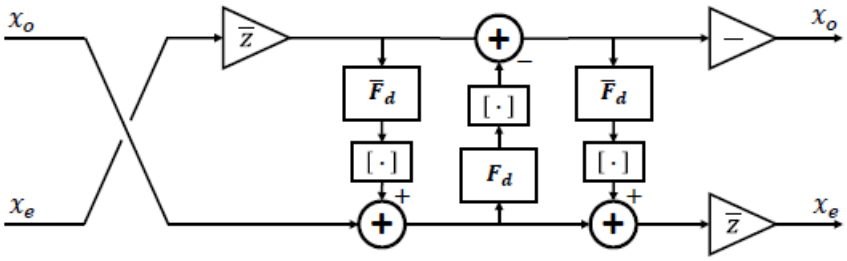
Con **Πj** e **Yj** essendo funzioni potenzialmente non lineari, e **[·]** è l'operatore di arrotondamento. Quando **IIj** e **Yj** sono lineari, e **aj, bj ∈ Z^n** sono vettori di lunghezza uguale, si può riscrivere sotto forma di matrix:

$$\begin{pmatrix} a_j \\ b_{j+1} \end{pmatrix} = \left[\begin{pmatrix} I_n & 0_n \\ \Pi_j & I_n \end{pmatrix} \begin{pmatrix} a_j \\ b_j \end{pmatrix} \right] \quad \text{and} \quad \begin{pmatrix} a_{j+1} \\ b_{j+1} \end{pmatrix} = \left[\begin{pmatrix} I_n & \Upsilon_j \\ 0_n & I_n \end{pmatrix} \begin{pmatrix} a_j \\ b_{j+1} \end{pmatrix} \right]$$

Dove **0_n** è un n x n matrix di zeri, **I_n** è una n x n identity matrix e **IIj**, **Yj** possono essere qualsiasi n x n matrix. Questa particolare forma matriciale è detta ***lifting matrix***. L'obiettivo è di fattorizzare la trasformata ricercata in un prodotto di *lifting matrix*, ottenendo così un'approssimazione intera di una trasformata definita su R o C.

Dalla matrice si può ottenere una decomposizione *lifting matrix* applicabile per qualsiasi *invertible matrix M*:

$$\begin{pmatrix} M & 0_n \\ 0_n & M^{-1} \end{pmatrix} = \begin{pmatrix} -I_n & 0_n \\ M^{-1} & I_n \end{pmatrix} \begin{pmatrix} I_n & -M \\ 0_n & I_n \end{pmatrix} \begin{pmatrix} 0_n & I_n \\ I_n & M^{-1} \end{pmatrix}$$



La figura rappresenta graficamente il *lifting scheme* [15] utilizzato per la *convolutional integer Fresnel transform*. Il *lifting scheme* è applicato su **odd(Xo)** e **even (Xe)** righe / colonne di un ologramma.

Quindi, metà del segnale sarà trasformato da M, e l'altra metà da M^{-1}. Questa proprietà è utile perché gli operatori di Fresnel sono unitari. Si può ridefinire la trasformata inversa di Fresnel in funzione della *forward Fresnel transform*.

Per la *convolutional form*, si ha

$$F_d^{-1} = \left(c\,\mathcal{F}^{-1}\,D^{-\lambda d/p^2}\,\mathcal{F} \right)^{-1} = \overline{c}\,\mathcal{F}^{-1}\,D^{\lambda d/p^2}\,\mathcal{F} = F_{-d} = \overline{F}_d$$

Questo significa che dopo la procedura di sollevamento, la prima parte del segnale sarà trasformata da **Fd**, e la seconda parte di **Fd coniugato**. Si prende il complesso coniugato sulla seconda parte del segnale per applicare una trasformata di Fresnel convolutional con distanza D su entrambi i segnali.

Per il caso *forma di Fourier*, si usa la seguente scomposizione:

$$\Psi_d = c\,D^{(p_d^2 - p_s^2)/\lambda d}\,\left(D^{p_s^2/\lambda d}\,\mathcal{F}\,D^{p_s^2/\lambda d} \right) = c\,D^{(p_d^2 - p_s^2)/\lambda d}\,\Phi_d$$

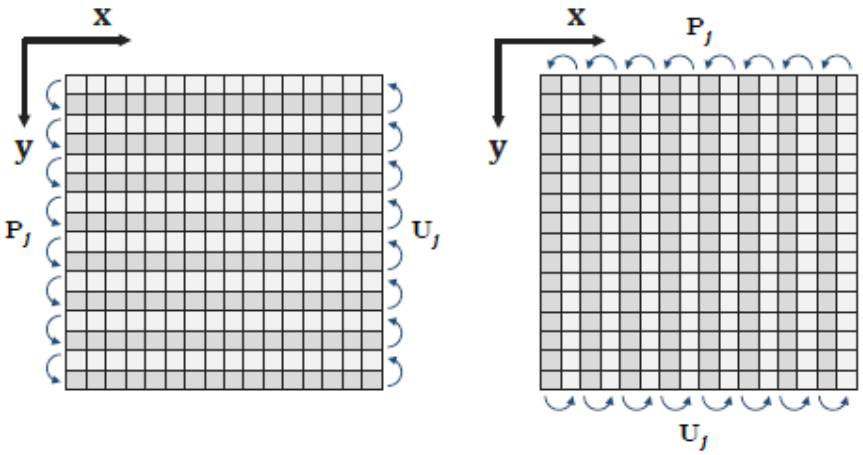
Dove

$$\Phi_d^{-1} = \left(D^{p_s^2/\lambda d}\,\mathcal{F}\,D^{p_s^2/\lambda d} \right)^{-1} = D^{-p_s^2/\lambda d}\,\mathcal{F}^{-1}\,D^{-p_s^2/\lambda d} = \Phi_{-d} = \overline{\Phi}_d$$

Si può trasformare il segnale per **Φd** con *lifting scheme* usando lo stesso approccio *convolution Fd*. Tuttavia, si deve $c\,D^{(p_d^2 - p_s^2)/\lambda d}$ moltiplicare con una funzione *pure phase delay*.

Per questo si può usare ***invertible integer lifting approximation della rotazione di Givens***. [16]

Dato che la trasformata di Fresnel è separabile, noi applichiamo la trasformata lungo le righe e colonne. Per ogni coppia consecutiva righe o colonne, le righe dispari Xo prediranno(Predict) le righe pari Xe, e le righe pari aggiorneranno(Update) le righe dispari nel lifting procedure.



(a) (b)

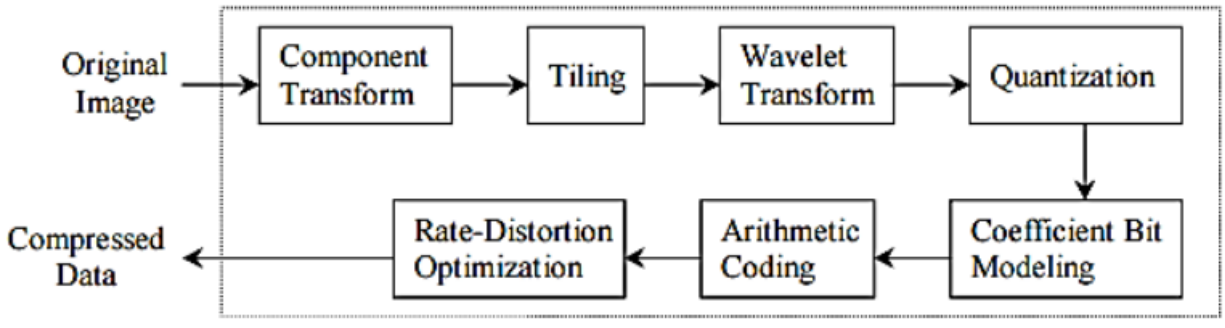
3.5 JPEG 2000

JPEG 2000 (JP2) è uno standard di compressione delle immagini e un sistema di codifica. È stato sviluppato dal 1997 al 2000 da un comitato del ***Joint Photographic Experts Group***, con l'intenzione di sostituire il loro standard JPEG basato sulla trasformazione del coseno discreto **DCT** con un metodo basato su ***wavelet***. [17]

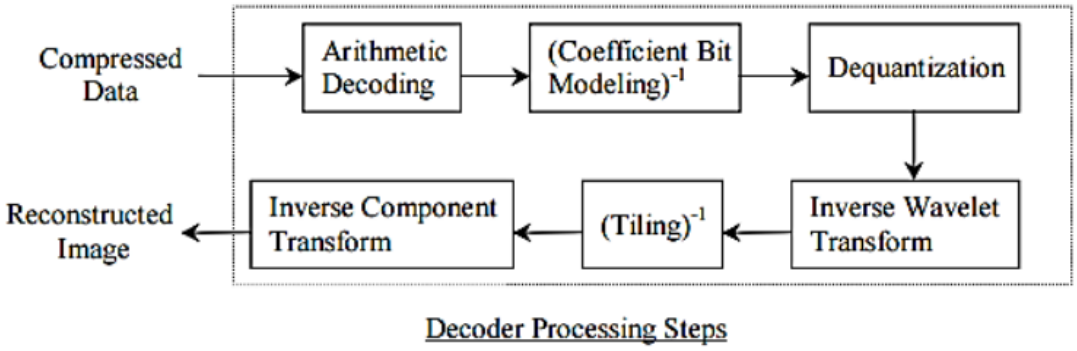
L'estensione del nome del file standardizzata è .jp2. JPEG 2000 è uno standard di compressione basato sulla trasformata *wavelet* discreta ***DWT*** che può essere adattato per la compressione video di immagini in movimento con l'estensione *Motion JPEG 2000*. La tecnologia *JPEG 2000* è stata scelta come standard di codifica video per il cinema.

Il principale vantaggio offerto da JPEG 2000 è la notevole flessibilità del *codestream*. Il *codestream* ottenuto dopo la compressione di un'immagine con JPEG 2000 è di natura scalabile, il che significa che può essere decodificato in diversi modi.

Le strutture semplificate *dell'encoder* e del *decoder* di JPEG-2000 sono mostrate nella Figura seguente. Supponiamo di avere un'immagine a più componenti. Le principali fasi di elaborazione *dell'encoder* sono: ***component transformation, tiling, wavelet transformation, quantization, coefficient bit modeling, arithmetic coding, e rate-distortion optimization.*** Il ruolo del decodificatore è quello di invertire i passaggi eseguiti dall'encoder, tranne il passaggio di *rate-distortion optimization step*.



Encoder Processing Steps



Decoder Processing Steps

3.6 Fasi JPEG 2000

3.6.1 Color components transformation

Inizialmente le immagini devono essere trasformate dallo spazio colore *RGB* in un altro spazio colore, portando a tre componenti che vengono gestiti separatamente. Ci sono due possibili scelte:

Irreversible Color Transform (ICT): utilizza il noto spazio colore YCBCR. Si chiama "irreversibile" perché deve essere implementato in float o fix-point e causa errori di arrotondamento.

Reversible Color Transform (RCT): utilizza uno spazio colore YUV modificato che non introduce errori di quantizzazione, quindi è completamente reversibile. La corretta implementazione dell'RCT richiede che i numeri siano arrotondati come specificato che non possono essere espressi esattamente in forma matriciale. La trasformazione è:

$$Y = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor; C_B = B - G; C_R = R - G;$$

$$\text{and } G = Y - \left\lfloor \frac{C_B + C_R}{4} \right\rfloor; R = C_R + G; B = C_B + G.$$

3.6.2 Tiling

Dopo la trasformazione del colore, l'immagine viene suddivisa nei cosiddetti riquadri, regioni rettangolari dell'immagine che vengono trasformate e codificate

separatamente. Le tessere possono essere di qualsiasi dimensione ed è anche possibile considerare l'intera immagine come una singola tessera. Una volta scelta la dimensione, tutte le tessere avranno la stessa dimensione (tranne opzionalmente quelle sui bordi destro e inferiore).

3.6.3 Trasformata wavelet

Queste tessere vengono quindi trasformate in *wavelet* a una profondità arbitraria, in contrasto con JPEG 1992 che utilizza una trasformata coseno discreta di dimensioni blocchi 8×8. JPEG 2000 utilizza due diverse trasformazioni wavelet:

1) **Irreversibile**: la trasformata wavelet CDF 9/7. Si dice "irreversibile" perché introduce rumore di quantizzazione che dipende dalla precisione del decoder.
2) **Reversibile:** una versione arrotondata della trasformata wavelet CDF 5/3 biortogonale. Utilizza solo coefficienti interi, quindi l'output non richiede arrotondamenti (quantizzazione) e quindi non introduce alcun rumore di quantizzazione.

Viene utilizzato nella codifica senza perdita di dati. Le trasformate wavelet sono implementate dallo schema di sollevamento o dalla convoluzione.

3.6.4 Quantization

Dopo la trasformata wavelet, i coefficienti vengono quantizzati scalare per ridurre il numero di bit per rappresentarli, a scapito della qualità. L'output è un insieme di numeri interi che devono essere codificati bit per bit. Il parametro che può essere modificato per impostare la qualità finale è il passo di quantizzazione: maggiore è il passo, maggiore è la compressione e la perdita di qualità. Con un passo di quantizzazione uguale a 1, non viene eseguita alcuna quantizzazione (viene utilizzata nella compressione senza perdita di dati).

3.6.5 Coding

Le sottobande quantizzate sono ulteriormente suddivise in distretti, regioni rettangolari nel dominio *wavelet*. Sono tipicamente selezionati in modo che i coefficienti al loro interno attraverso le sottobande formano blocchi spaziali approssimativamente nel dominio dell'immagine (ricostruito), sebbene questo non sia un requisito. I distretti sono ulteriormente suddivisi in blocchi di codice. I blocchi di codice si trovano in una singola sottobanda e hanno dimensioni uguali, tranne quelli situati ai bordi dell'immagine. L'encoder deve codificare i bit di tutti i coefficienti quantizzati di un blocco di codice, partendo dai bit più significativi e procedendo ai bit meno significativi mediante un processo chiamato schema ***EBCOT*** (*Embedded Block Coding with Optimal Truncation*).

3.6.6 Compression ratio

Rispetto al precedente standard JPEG, *JPEG 2000 offre un guadagno di compressione tipico nell'intervallo del 20%, a seconda delle caratteristiche dell'immagine*. Le immagini a risoluzione più elevata tendono a trarre maggiori benefici, laddove la previsione della ridondanza spaziale di JPEG-2000 può contribuire maggiormente al processo di compressione. In applicazioni a bit rate molto basso, gli studi hanno dimostrato che JPEG 2000 ha prestazioni migliori della modalità di codifica intra-frame di H.264. Buone applicazioni per JPEG 2000 sono immagini di grandi dimensioni, immagini con bordi a basso contrasto, ad esempio immagini mediche.

3.6.7 Computational complexity and performance

JPEG2000 è molto più complicato in termini di *complessità computazionale* rispetto allo standard JPEG. La tiling, la trasformazione del componente colore, la trasformazione wavelet discreta e la quantizzazione potrebbero essere eseguite abbastanza velocemente, sebbene il codec entropico richieda tempo e sia piuttosto complicato. La modellazione del contesto **EBCOT** e il codificatore MQ aritmetico richiedono la maggior parte del tempo del codec JPEG2000.

Sulla CPU l'idea principale di ottenere una rapida codifica e decodifica JPEG 2000 è strettamente connessa con AVX/SSE e multithreading per elaborare ogni riquadro in thread separati. Le soluzioni JPEG 2000 più veloci utilizzano la potenza sia della CPU che della GPU per ottenere benchmark ad alte prestazioni.

3.7 Vantaggi JPEG 2000

JPEG 2000 offre i seguenti vantaggi rispetto JPEG: [17]

- La capacità di mostrare dati a due livelli, in scala di grigi, a colori e a colori in diversi spazi colore.
- Nessun limite per la quantità di informazioni private o per scopi speciali nei metadati.
- Estensibilità ed evoluzione senza soluzione di continuità in base alle esigenze di nuove funzionalità.
- Una scelta di schemi di compressione con compromessi spazio-temporali per gli sviluppatori di applicazioni.
- La capacità di gestire immagini di grandi dimensioni, ovvero quelle maggiori di 64k x 64k pixel, anche immagini naturali e generate al computer, senza affiancamento.
- Compressione a bassa velocità in bit fino a meno di 0,25 bit per pixel per immagini in scala di grigi ad alta risoluzione.

3.8 Svantaggi JPEG 2000

Per quanto eccezionale sia JPEG 2000, gli effetti indesiderati dell'adozione di esso non sono banali, da qui l'uso e il supporto limitati del formato. [17]

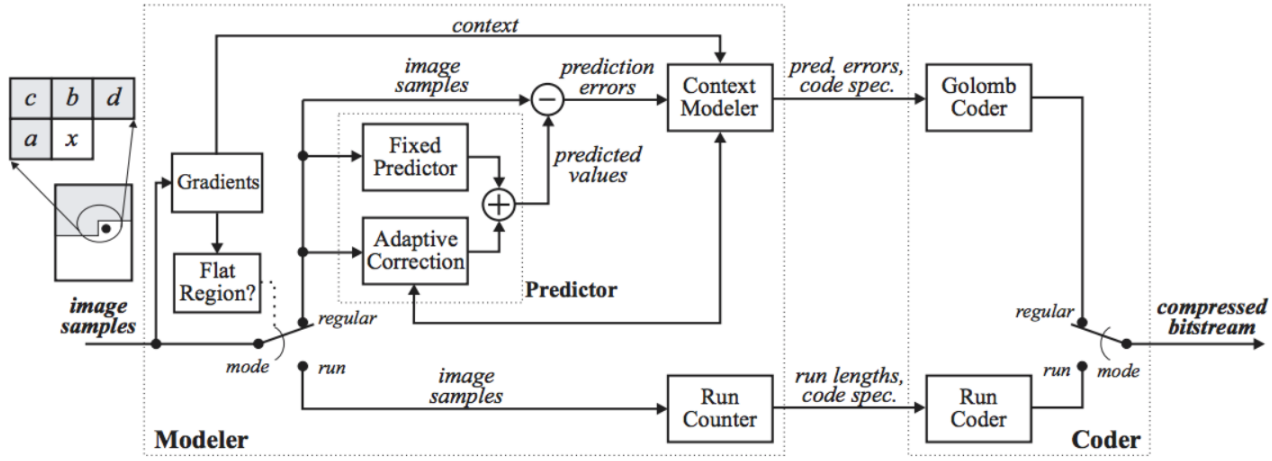
- JPEG 2000 funziona solo in determinati browser.
- JPEG 2000 non è compatibile con le versioni precedenti. Per sfruttare anche il formato JPEG originale, gli sviluppatori e altri professionisti digitali che adottano JPEG 2000 devono codificare in un nuovo standard.
- La codifica dei file JPEG 2000, che comporta la modifica della CPU e l'aggiunta di codice, richiede molte risorse e richiede molta più memoria per l'elaborazione. Data l'elevata capacità delle macchine moderne, questo problema probabilmente non esiste più. Tuttavia, quando JPEG 2000 ha debuttato per la prima volta nel 2000, il suo requisito di memoria ha rappresentato una preoccupazione significativa.
- I siti Web e i produttori di fotocamere hanno ritardato l'accettazione di JPEG 2000 fino a quando non è stato ampiamente adottato.
- Il processo di codifica di JPEG 2000 è più lento e complicato di quello di JPEG.
- JPEG 2000 è molto meno adattabile ai contenuti rispetto a JPEG e la scelta di un bit rate di destinazione estremamente basso rovinerebbe l'immagine. A seconda del contenuto dell'immagine, dovrai regolare manualmente il bit rate.

3.9 JPEG-LS

JPEG-LS è uno standard di compressione lossless per immagini a tono continuo. È un algoritmo di base semplice ed efficiente che consiste in due fasi indipendenti e distinte chiamate modellazione e codifica. JPEG-LS è stato sviluppato con l'obiettivo di fornire uno standard di compressione delle immagini *lossless* e quasi *lossless* a bassa complessità in grado di offrire una migliore efficienza di compressione rispetto al JPEG lossless. È stato sviluppato perché all'epoca lo standard *JPEG lossless* basato sulla codifica di Huffman e altri standard erano limitati nelle loro prestazioni di compressione. Il nucleo di JPEG-LS si basa sull'algoritmo **LOCO-I (HP),** che si basa sulla previsione, sulla modellazione dei residui e sulla codifica dei residui basata sul contesto.

La maggior parte della bassa complessità di questa tecnica deriva dal presupposto che i residui di previsione seguano una distribuzione geometrica bilaterale (chiamata anche distribuzione discreta di Laplace) e dall'uso di ***Golomb*** codici simili, noti per essere approssimativamente ottimali per le distribuzioni geometriche. Oltre alla compressione senza perdita, JPEG-LS fornisce anche una modalità con perdita ("*near-lossless*") in cui l'errore assoluto massimo può essere controllato *dall'encoder*.

L'algoritmo JPEG-LS esamina alcuni vicini del pixel corrente e li utilizza come contesto che verrà utilizzato per predire il valore di quel pixel e per selezionare una distribuzione di probabilità che verrà usata per codificare la differenza di predizione rispetto al valore del pixel, utilizzando *Golomb*. Inoltre, è presente una modalità di funzionamento chiamata ***run mode***, che viene utilizzata quando sono presenti più pixel che hanno lo stesso valore e permette di codificare la lunghezza di questa "lista" di pixel uguali piuttosto che codificarli uno ad uno.
Il contesto definito per codificare un pixel **X** è l'insieme dei pixel a sinistra di X (**W**), in alto a sinistra **NW**, sopra **N** e in alto a destra **NE**. Il codificatore esamina i pixel del contesto e decide se codificare il pixel X in ***run mode*** **oppure in** ***regular mode.***



- Run mode: Si conta la lunghezza della sequenza e tramite il run coder si codifica questo valore. Infine, si invia al decompressore, che sa di trovarsi in run mode e decodifica il valore, inviandolo in output.

- Regular mode Viene calcolata una predizione del valore di X tramite il *fixed predictor*, che utilizza una funzione chiamata ***Median Edge Detector (MED):***

$$\hat{x}_{MED} = min(I_w, I_n, I_{nw}) + max(I_w, I_n, I_{nw}) - I_{nw}$$

Una volta calcolato questo valore, tramite *l'adaptive correction* si ottiene una correzione con un termine che dipende dal contesto, che viene codificato tramite la codifica di *Golomb* che utilizza un parametro sintonizzabile M per dividere un valore di input x in due parti: **q**, il risultato di una divisione per M, e **r**, il resto. Il quoziente viene inviato in codifica unaria, seguito dal resto in codifica binaria troncata.

Per codificare si utilizza un modello di probabilità che evita il problema della ***context dilution***, ovvero cerca di creare meno contesti possibili, attraverso l'utilizzo di ***gradients***, che stimano il valore da dare al codice di ***Golomb***. Il codice di Golomb poi dipende da tutti e quattro i pixel del contesto e dall'errore di predizione che era stato codificato precedentemente per lo stesso contesto.

3.10 Prestazioni

La compressione per JPEG-LS è generalmente molto più veloce di JPEG 2000 e molto migliore dello standard JPEG lossless originale. La compressione per JPEG-LS è generalmente leggermente peggiore di JPEG 2000, ma questo dipende fortemente contenuto dell'immagine.
La decompressione per JPEG-LS è generalmente molto più veloce di JPEG 2000, ma ciò dipende dalle esatte implementazioni del decodificatore software. Questa dichiarazione è valida per le implementazioni open source.

Capitolo 4

4. Implementazione

4.1 Hardware e Software

Per la realizzazione di questo lavoro sono stati utilizzati il linguaggio Python (3.9.0). In particolare, sono state sfruttate le seguenti librerie: Numpy (1.21.4), Matplotlib (3.5.1), Scipy (1.7.3), Opencv (4.5.4.60).

La macchina su cui è stato svolto il lavoro è stata un XMG Fusion 15, Intel Core i7-9750H 2.60 GHz hexa-core burst 4.5 GHz con sistema operativo Windows 11.

Non è stata realizzata un'interfaccia grafica, in quanto il progetto prende in input un file di testo formattato nel seguente modo:

```
exampleholo.mat,8e-6,532e-9,5e-1,holo,conv,0
```

Il primo campo di ogni riga contiene il nome del file. mat che al suo interno contiene l'ologramma, nel caso in cui non si ha a disposizione direttamente ologramma.

Il secondo campo contiene la dimensione del pixel pitch, successivamente si ha la dimensione wavelenght, poi si ha propagation depth, campo del nome dell'ologramma holo in questo caso, l'algoritmo utilizzato per la trasformata di Fresnel forma convolutionale o fourier. Ultimo campo contiene un flag per effettuare compressione e decompressione (0), comprimere soltanto 1, decomprimere 2.

4.2 Hologram Compression

In questa sezione, si mostra il file HologramCompression.py, che ha la funzione main.

```
algCompression="JPEG-LS" # JPEG_2000 o JPEG-LS
if (algCompression== "JPEG_2000"):
    ext=".jp2"
elif (algCompression=="JPEG-LS"):
    ext=".jls"
else:
    ext=""
path="./input.txt"
file_output=open("./output_"+algCompression+".txt", "a")
file_input=open(path,"r")
while(i := file_input.readline()):
    input=i.split(",")
    #print(input)
    start=time.process_time()
    f = scipy.io.loadmat(input[0])  # aprire il file .mat
    #f=mat73.loadmat(input[0]).get('CGH') #caso file holo interno alla struttura CGH
    nome=input[0].replace(".mat","")
    pp = input[1]  # pixel pitch
    pp = np.matrix(pp)
    wlen = input[2]  # wavelenght
    wlen = np.matrix(wlen)
    dist = input[3] # propogation depth
    dist = np.matrix(dist)
    nameOlogram=input[4]
    algoritmo=input[5] #four o conv
    flag=input[6].replace("\n","") #0 comprime e decomprime, 1 comprime soltanto, 2 decomprime soltanto
```

Nella prima parte del main andiamo a settare quale algoritmo di compressione utilizzare tra JPEG 2000 e JPEG-LS, dopodiché andiamo a leggere dal file input.txt i valori per ogni ologramma (pixel pitch, wavelenght, propagation depth, nome dell'ologramma, conv o four e flag). Prima di tutto si apre il file .mat contenente l'ologramma attraverso la funzione loadmat di scipy o nel caso in cui il file .mat all'interno ha le informazioni dell'ologramma in sottoforma di una struttura utilizziamo la funzione loadmat di mat73. Tutte le informazioni dell'ologramma vengono passate come parametro alla funzione np.matrix di numpy che restituisce una matrice.

Nel main dopo aver preso in input i parametri che occorrono, si ha il processo di compressione che applica la trasformata di Fresnel utilizzando uno tra Fourier e Convolutional form.

```
#processo per la compressione
if flag == 0 or 1:
    holo = np.matrix(f.get(nameOlogram))  # estraggo dal file .mat solo holo e lo converto da np a array
    algo = algoritmo#conv o four
    t = intfresnel2D(np.csingle(holo), False, pp, dist, wlen, algo)
    if algo=="four":
        t = fourier_phase_multiply(t, False, pp, dist, wlen)
    if (algCompression == "JPEG 2000"):
        cv2.imwrite(nome + "_" + algo + ext, np.real(t))
    elif (algCompression == "JPEG-LS"):
        img = Image.fromarray(np.uint8(np.real(t)))
        img.save(nome + "_" + algo + ext)
```

Per prima cosa estraggo dal file .mat ologramma e lo converto in un array 2D (matrice). Dopodiché si chiama la funzione intfresnel2D che effettua la trasformata di Fresnel (convolutiva o Fourier) in base al parametro “algo” passato in input, se algoritmo usato è Fourier l’output di intFresnel2D viene passato in input alla funzione fourier_phase_multiply, restituendo la matrice che verrà compressa tramite openCV in JPEG 2000 oppure tramite Pillow in JPEG-LS però prima occorre trasformare la matrice come un oggetto Pillow. Se invece “algo” in input è convolutiva allora viene chiamata solamente la funzione intfresnel2D e la compressione avviene allo stesso modo come nel caso di Fourier.

All’interno della funzione main c’è anche il processo di decompressione:

```
#processo decompressione
if flag==0 or 2:
    start=time.time()
    if (algCompression == "JPEG 2000"):
        holo = cv2.imread(nome+"_"+algo+ext,0)
    elif (algCompression == "JPEG-LS"):
        holo = Image.open(nome + "_" + algo + ext)
    else:
        break;
    if(algo=="conv"):
        holo = np.matrix(holo)
    algo = algoritmo  # conv o four
    t = intfresnel2D(np.csingle(holo), False, pp, dist, wlen, algo)
    if algo == "four":
        t = fourier_phase_multiply(t, False, pp, dist, wlen)
```

Semplicemente fa l'inverso, partendo dall'immagine compressa, decomprime con l'opportuno algoritmo di decompressione JPEG 2000 o JPEG-LS e applica la trasformata di Fresnel inversa dopodiché se algo è fourier si ha bisogno di chiamare la funzione fourier_phase_multiply, altrimenti se "algo" è convolutiva si ha bisogno di chiamare solamente intfresnel2D così da ottenere nuovamente l'ologramma di partenza, ciò si può vedere nel capitolo 5 dove vengono mostrati degli esempi.

4.3 Fresnel Implementation

All'interno della funzione main viene chiamata la funzione intfresnel2D che applica una versione intera della trasformata di Fresnel 2D per gli ologrammi con dimensioni quadrate, utilizzando lo schema di sollevamento generalizzato. In input riceve: segnale intero complesso x, direzione booleana, pixel pitch, profondità di propagazione, lunghezza d'onda e algoritmo scelto. Lunghezze sono espresse in metri.

```
def intfresnel2D(x, fw, pp, z, wlen, algo):

    assert (np.size(x) % 2) == 0, "Hologram dimensions must be even"
    assert np.size(x, 0) == np.size(x, 1), "Hologram must be square (to be
resolved in next version)"

    #se conv usa le prime due
    def fw_fresnel(r):
        return np.round(fpropfun(r, pp, fz, wlen))

    def bw_fresnel(r):
        return np.round(bpropfun(r, pp, bz, wlen))

    # se fourier usa
    def fwdfourierfresnel1D(x, pp, z, wlen):
        return fourierfresnel1D.fourierfresnel1D(x, pp, z, wlen, True)

    def revfourierfresnel1D(x, pp, z, wlen):
        return fourierfresnel1D.fourierfresnel1D(x, pp, z, wlen, False)

    def apply_transform():
        if (fw):
            o = np.conjugate(x[:, 1::2])
            e = x[:, 0::2]

            e = e + bw_fresnel(o)
            o = o - fw_fresnel(e)
            e = e + bw_fresnel(o)

            x[:, 0::2] = -o
            x[:, 1::2] = np.conjugate(e)
```

```
            return x
        else:
            e = np.conj(x[:, 1::2])  # righe le prende tutte, colonne prende a 2
a 2
            o = -x[:, 0::2]  # righe le prende tutte, colonne parte dalla prima
e a step di 2

            e = e - bw_fresnel(o)
            o = o + fw_fresnel(e)
            e = e - bw_fresnel(o)

            x[:, 0::2] = e
            x[:, 1::2] = np.conj(o)
            return x

    if algo == "conv":
        fpropfun = confresnel1D.convfresnel1D  # convfresnel1D(x,pp,z,wlen)
        bpropfun = confresnel1D.convfresnel1D  # convfresnel1D(x,pp,z,wlen)
        fz = z
        bz = -z
    elif algo == "four":
        fpropfun = fwdfourierfresnel1D
        fz = z
        bpropfun = revfourierfresnel1D
        bz = z
    else:
        print("algoritmo sconosciuto")

    for i in [-1, 1]:
        if fw:
            x = np.rot90(x, i)
        x = apply_transform()
        if not fw:
            x = np.rot90(x, i)
    return x
```

La funzione intfresnel2D per prima cosa controlla se la dimensione dell'ologramma è pari (np.size ritorna il prodotto righe per colonne), perché non si potrebbe applicare in modo efficiente il lifting scheme.

Se algoritmo usato è convolutional allora setta come Fw_fresnel con la funzione convfresnel1D, e Bw_Fresnel sempre la stessa funzione. Mentre se l'algoritmo usato è fourier allora come Fw_fresnel utilizza la funzione fwdfourierfresnel1D, mentre come Bw_fresnel utilizza revfourierfresnel1D. Dopo aver fatto queste operazioni preliminari si applica il lifting scheme, come illustrato nel capitolo precedente, sulle colonne dispari e pari applicando la trasforma di Fresnel (apply_trasform) mostrata in figura utilizzando fw_fresnel e bw_fresnel settati in modo opportuno con la forma convolutiva oppure Fourier.

```
def apply_transform():
    if (fw):
        o = np.conjugate(x[:, 1::2])
        e = x[:, 0::2]

        e = e + bw_fresnel(o)
        o = o - fw_fresnel(e)
        e = e + bw_fresnel(o)

        x[:, 0::2] = -o
        x[:, 1::2] = np.conjugate(e)

        return x
    else:
        e = np.conj(x[:, 1::2])  # righe le prende tutte, colonne prende a 2 a 2
        o = -x[:, 0::2]  # righe le prende tutte, colonne parte dalla prima  e a
step di 2

        e = e - bw_fresnel(o)
        o = o + fw_fresnel(e)
        e = e - bw_fresnel(o)

        x[:, 0::2] = e
        x[:, 1::2] = np.conj(o)
        return x
```

La funzione apply_transform per prima cosa controlla se fw = true, prende le colonne partendo dalla prima, a step di 2(colonne dispari) le salva in “o” applicando la funzione numpy conjugate che restituisce il coniugato complesso, cioè cambia il segno della sua parte immaginaria. Mentre in “e “si ha le colonne pari. E su questi valori si applica la trasformata di Fresnel. Mentre se fw = false allora quello che si fa è il procedimento inverso, cioè sulle colonne pari si effettua la funzione conjugate e mentre sulle colonne dispari ai valori si ha un cambio di segno, dopo aver fatto questo si applica la trasformata di Fresnel. Vediamo in seguito le funzioni confresnel1D e FourierFrensel1D.

In confresnel1D.py si trovano le funzioni applicate durante Fresnel con convolutional form.

```
def convfresnel1D(x, pp, z, wlen):
    c = -1j * wlen * z * math.pi
    n = x.shape[0]
    w = np.matrix(np.arange(-n / 2, n / 2)).getH() / (np.matmul(np.matrix(n),
pp))
    potenza = np.power(w, 2)
    molt = c[0, 0] * potenza
    espon = np.exp(molt)
    y = IDFT((np.multiply(DFT_slow(x), np.fft.ifftshift(espon))))
    return y
```

La funzione convfresnel1D calcola la trasformata di Fresnel convolutiva unitaria secondo la formula sottostante:

$$F_d = c \cdot \mathcal{F}^{-1} \cdot D^{-\lambda d/p^2} \cdot \mathcal{F}$$

Dove F è una matrice DFT nxn computata usando FFT (Fast Fourier Trasform), c=exp(ikd) e D è la diagonal matrix con un profilo di fase quadratico e ampiezza unitaria, definita come:

$$D = \mathrm{diag}\left(\exp\left(i\pi(\tfrac{x}{n})^2\right) \right) \quad \text{for} \quad x \in \left\{ -\frac{n}{2}, -\frac{n}{2}+1, \ldots, \frac{n}{2}-1 \right\}$$

Per un segnale composto da n campioni.

È stata omessa la divisione costante $i\lambda d$ aggiuntiva di c da dove è presente nella definizione tipica espressa nel sotto capitolo 3.2

Questo viene fatto per ottenere un intervallo dinamico simile nel object plane rispetto al piano dell'ologramma per l'efficienza di codifica delle trasformate intere di Fresnel.

Nella funzione convfresnel1D utilizziamo rispettivamente la trasformata Fourier e la trasformata inversa di Fourier.

Nella figura seguente viene implementata la trasformata di Fourier applicata per ciascuna colonna

```
def DFT_slow(x):
    """Compute the discrete Fourier Transform of the 1D array x"""
    x = np.matrix(x)
    N = x.shape[0]
    n = np.arange(N)
    k = n.reshape((N, 1))
    M = np.exp(-2j * np.pi * k * n / N)
    return np.dot(M, x)
```

mentre nella figura seguente è la formula per calcolare la trasformata di Fourier

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,k\,n\,/\,N}$$

Nella figura seguente invece viene implementata la trasformata Fourier inversa

```
def IDFT(x):
    x = np.matrix(x)
    N = x.shape[0]
    n = np.arange(N)
    k = n.reshape((N, 1))
    M = np.exp(2j * np.pi * k * n / N)
    molt = np.dot(M, x)
    return molt / N
```

Mentre nella figura, in seguito, è la formula per calcolare la trasformata Fourier inversa

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i\,2\pi\,k\,n\,/\,N}$$

Nel caso l'algoritmo scelto sia Fourier tutte le funzioni si trovano in fourierfresnel1D.py.

```
def fourierfresnel1D(x, pp, z, wlen, forward):
    c = 1j * np.pi / wlen / z
    if not forward:
        c = -c
    n = np.size(x, 0)

    w = np.matrix(np.arange(-n / 2, n / 2)).getH()
    e = np.exp(c[0,0] * (np.power((w * pp), 2)))

    if forward:
        x = np.fft.fftshift(np.multiply(x, e), 0)
        y = np.multiply(e, np.fft.fftshift(DFT_slow(x), 0)) / np.sqrt(n)
        return y
    else:
        mult = np.multiply(e,x)
        x = IDFT(np.fft.ifftshift(mult,0))
        y = np.sqrt(np.size(x, 0)) * np.multiply(np.fft.ifftshift(x, 0), e)
        return y
```

La funzione fourierfresnel calcola la trasformata di Fresnel con la forma di Fourier.

Il parametro input forward è un flag che esegue la funzione in modalità forward o inversa.

La forma di fourier cambia il pixel pitch di destinazione che dipende dalla lunghezza d'onda e la distanza di propagazione.

$$\Psi_d = c \cdot D^{p_d^2/\lambda d} \cdot \mathcal{F} \cdot D^{p_s^2/\lambda d} \qquad p_d = \frac{p_s}{\lambda d};$$

dove D è dato da

$$D = \mathrm{diag}\left(\exp\left(i\pi \left(\frac{x}{n}\right)^2 \right) \right) \qquad x \in \left\{ -\frac{n}{2}, -\frac{n}{2}+1, \ldots, \frac{n}{2}-1 \right\}$$

Nella forma di Fourier, si usa la seguente decomposizione:

$$\Psi_d = c\, D^{(p_d^2 - p_s^2)/\lambda d} \left(D^{p_s^2/\lambda d}\, \mathcal{F}\, D^{p_s^2/\lambda d} \right) = c\, D^{(p_d^2 - p_s^2)/\lambda d}\, \Phi_d$$

Dove si ha

$$\Phi_d^{-1} = \left(D^{p_s^2/\lambda d}\, \mathcal{F}\, D^{p_s^2/\lambda d} \right)^{-1} = D^{-p_s^2/\lambda d}\, \mathcal{F}^{-1}\, D^{-p_s^2/\lambda d} = \Phi_{-d} = \overline{\Phi}_d$$

Adesso si ha bisogno di trasformare Φd con il lifting scheme usando lo stesso approccio della forma convolutional. Bisogna moltiplicare con un ***pure phase delay function (fourier_phase_multiply).*** Per questo, si utilizza invertible integer lifting approssimazione di *Givens (givenphaserot*).

```
def fourier_phase_multiply(r, fw, pp, z, wlen):
    n = np.size(r, 0)
    xx = np.power(np.matrix(np.arange(-n / 2, n / 2)), 2)
    ppout = wlen * np.abs(z) / n / pp

    temp = (math.pi * np.power(ppout, 2)) / ((wlen * np.abs(z)))
    p = np.multiply(temp, (xx + xx.transpose())) + (2 * np.pi * z) / wlen
    return givenphaserot(r, p, fw)
```

```
def givenphaserot(x, p, fw):
    assert np.size(x) == np.size(p), "Input argument X & P must be equal
dimension"

        q = np.mod(np.floor(2*p/np.pi+0.5), 4)
        t = np.mod(p+(np.pi/4),np.pi/2) - (np.pi/4)


        peps = 1e-7  # phase epsilon
        if fw:
            x = quadrant_swap(x,q)

        x = givens_rot(t, x)

        if not fw:
            x = quadrant_swap(x,np.mod(4 - q, 4))
        return x
```

```
# givens rotation
def givens_rot(theta, x):
    m = np.abs(theta) < peps

    a = np.divide((np.cos(theta) - 1), np.sin(theta))
    a[m] = 0

    b = np.sin(theta)
    b[m] = 0

    if fw:
        sg = 1
    else:
        sg = -1

    x = x + sg * np.round(np.multiply(a, np.imag(x)))

    x = x + sg * 1j * np.round(np.multiply(b, np.real(x)))

    x = x + sg * np.round(np.multiply(a, np.imag(x)))

    return x
```

```
# determine in what phase quadrant  to operate
def quadrant_swap(xtemp, q):
    m = np.logical_or(q == 1, q == 3)
    np.putmask(xtemp, m, (-np.imag(xtemp)) + (1j * np.real(xtemp)))
    np.putmask(xtemp, q > 1, -xtemp)
    return xtemp
```

Capitolo 5

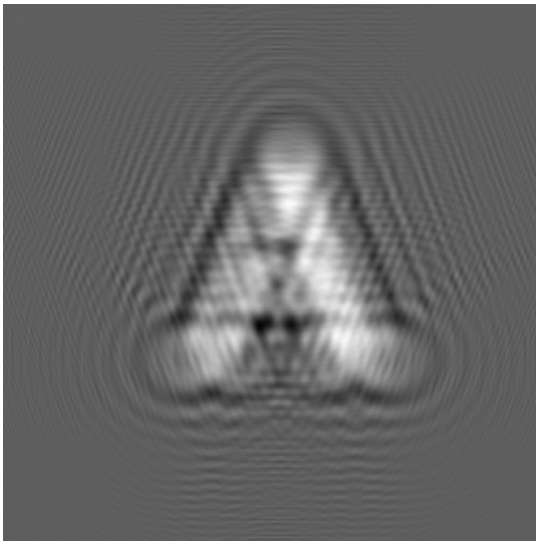
5. Esempi di compressione e decompressione

In questo capitolo viene mostrato degli esempi di esecuzione del codice. In particolare, si farà vedere la compressione e decompressione dell'ologramma utilizzando la trasformata di Fresnel (convolutiva e Fourier) con l'aggiunta dell'algoritmo di compressione JPEG200 e JPEG-LS.

5.1 Compressione JPEG 2000

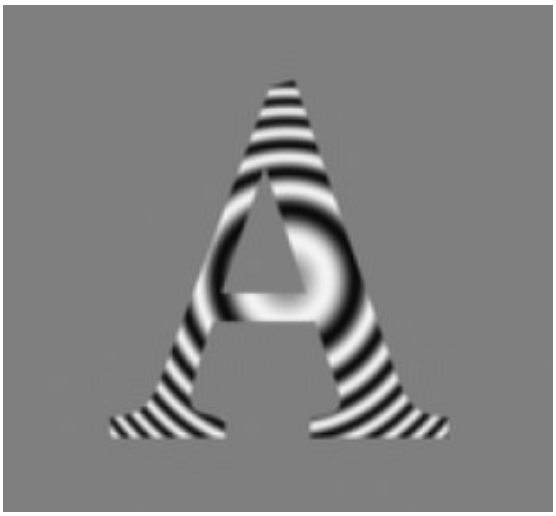
In questa sezione viene mostrato come cambia l'ologramma nelle varie fasi della compressione.

In questa l'immagine mostra l'ologramma prima di effettuare la compressione.



In queste immagini invece mostra l’ologramma dopo aver applicato la trasformata di Fresnel in forma convolutiva e Fourier.

Forma Convolutiva

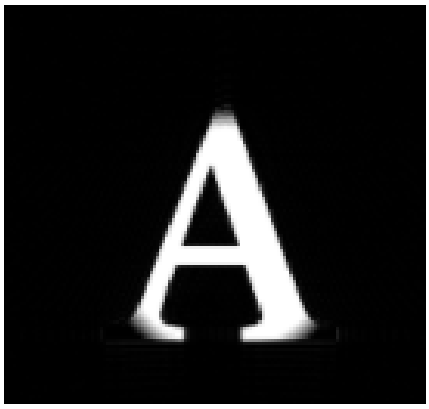


Forma Fourier

In questa è immagine finale viene mostrato l’ologramma dopo aver utilizzato la trasformata di Fresnel in forma convolutiva e l’algoritmo di compressione JPEG 2000.

Per la trasformata di Fresnel in forma Fourier si ha bisogno di un ulteriore processo di effettuare una point-wise multiplication con un ritardo di fase e in questa immagine viene mostrato l’ologramma dopo aver effettuato ciò.

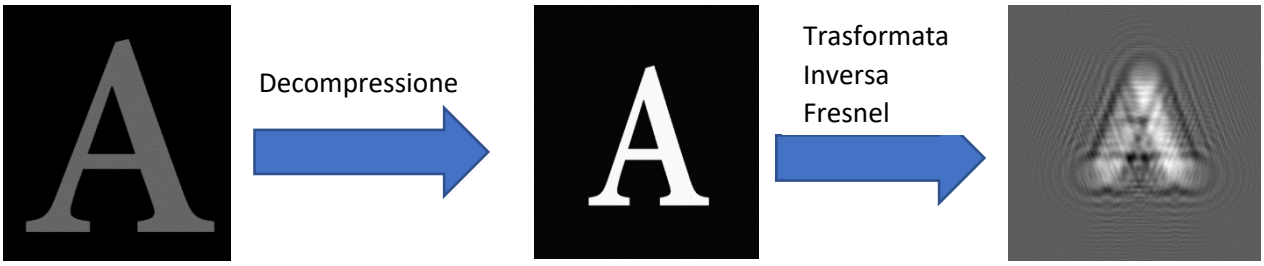
In questa immagine finale viene mostrato l'ologramma dopo aver utilizzato la trasformata di Fresnel in forma Fourier e l'algoritmo di compressione JPEG 2000.



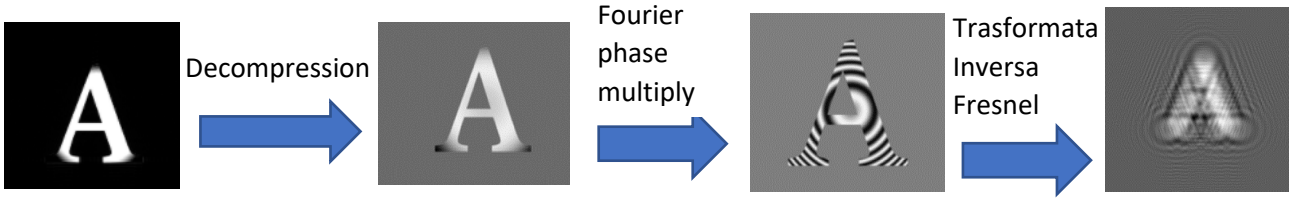
5.2 Decompressione JPEG 2000

Nella decompressione il procedimento è lo stesso ma viene fatto in maniera inversa, cioè prima l'ologramma viene decompresso con l'algoritmo jpeg 2000 e poi viene applicata la trasforma inversa di Fresnel.

In questa serie di immagini è si può notare come cambia l'ologramma durante il processo di decompressione utilizzando la trasformata di Fresnel in forma convolutiva.



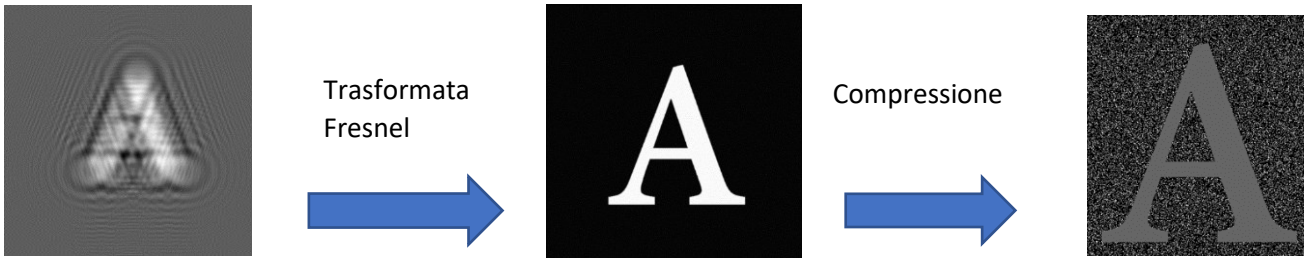
In questa serie di immagini è si può notare come cambia l'ologramma durante il processo di decompressione utilizzando la trasformata di Fresnel in forma Fourier.



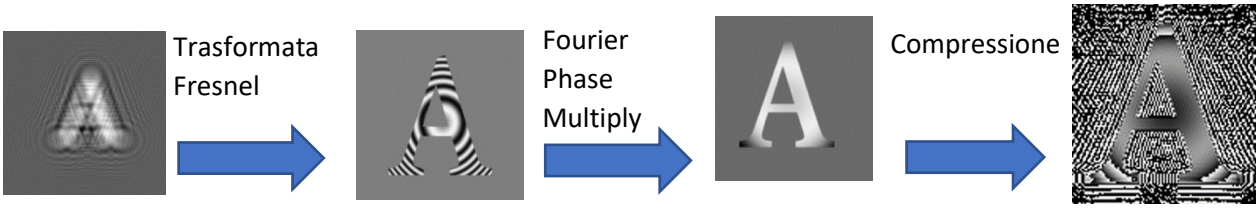
5.3 Compressione JPEG-LS

In questa sezione viene mostrato come cambia l'ologramma nelle varie fasi della compressione utilizzando l'algoritmo di compressione JPEG-LS. Ovviamente la sequenza degli ologrammi è identica cambia solamente l'immagine finale.

In questa serie di immagini è si può notare come cambia l'ologramma durante il processo di compressione utilizzando la trasformata di Fresnel in forma convolutiva.



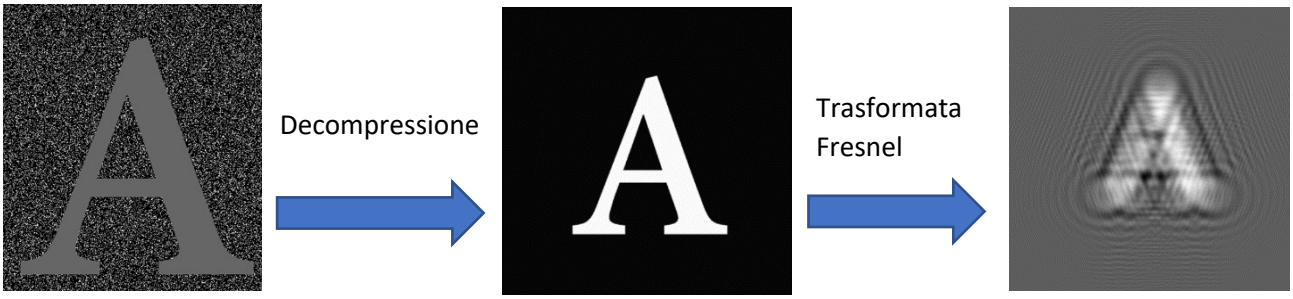
In questa serie di immagini è si può notare come cambia l'ologramma durante il processo di compressione utilizzando la trasformata di Fresnel in forma Fourier.



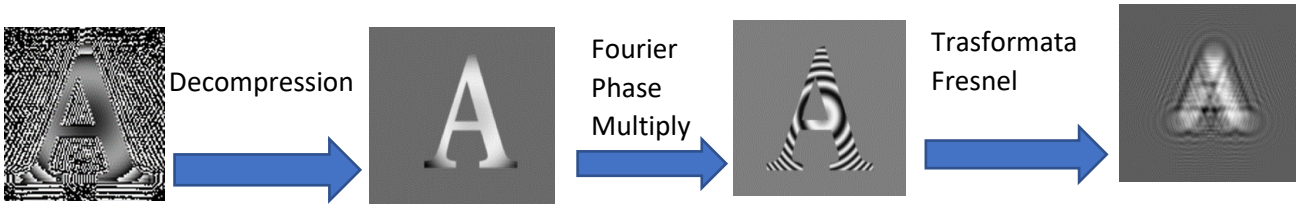
5.4 Decompressione JPEG-LS

In questa sezione viene mostrato come cambia l'ologramma nelle varie fasi della decompressione utilizzando l'algoritmo di decompressione JPEG-LS. Ovviamente, la sequenza di immagine è la stessa ma in maniera inversa.

In questa serie di immagini è si può notare come cambia l'ologramma durante il processo di decompressione utilizzando la trasformata di Fresnel in forma convolutiva.



In questa serie di immagini è si può notare come cambia l'ologramma durante il processo di decompressione utilizzando la trasformata di Fresnel in forma Fourier.



Capitolo 6

6. Risultati sperimentali

6.1 Dataset

Per la realizzazione di questo lavoro sono stati utilizzati 6 ologrammi presi da 2 diversi database: il database Interfere-II [18] e il database EmergImg-HoloGrail v2. [19]

La tabella riporta tutti i principali parametri dell'ologramma, risoluzione, lunghezza d'onda (λ) , pixel pitch (p) e la destination plane (d) scelta.

Name	Database	Form Resolution	(pixels)	λ (nm)	p (µm)	d (mm)
Ball	Interfere-II	Conv.	8192 × 8192	633.0	1.0	140.0
Dragon	Interfere-II	Conv.	8192 × 8192	633.0	1.0	140.0
Venus	Interfere-II	Conv.	8192 × 8192	633.0	1.0	145.0
Astronauts	EmergImg-HoloGrail	Fourier	2588 × 1940	632.8	2.2	172.1
Dices 2	EmergImg-HoloGrail	Fourier	2588 × 1940	632.8	2.2	159.5
Skull	EmergImg-HoloGrail	Fourier	2588 × 1940	632.8	2.2	168.9

6.2 Analisi dei risultati

Per l'analisi dei risultati si è adottato un approccio simile agli ideatori del paper "Integer Fresnel Transform for Lossless Hologram Compression". Gli autori avevano testato solo JPEG 2000 e JPEG 2000 dopo applicazione della trasformata intera di Fresnel.

Nel nostro caso andremo a testare il dataset attraverso JPEG 2000 dopo applicazione della trasformata intera di Fresnel e JPEG-LS dopo applicazione della trasformata di Fresnel.

La prima fase dell'analisi ha come obiettivo di indicare quale dei due algoritmi di compressione sia più performante per dimensione del file compresso e tempi di compressione e decompressione. Per avere un resoconto esaustivo per i tempi di compressione e decompressione i test sono stati eseguiti 3 volte per ciascuno ologramma in modo da ottenere una media dei valori.

6.3 Test JPEG 2000

Nome	Tempo medio compressione	Tempo medio decompressione	Dimensione iniziale	Dimensione compressa	Metodo
exampleholo	1,58221	1,54800	1176	164	Convolutional
ball8KS	178,75346	176,31408	493532	8431	Convolutional
dragon8KS	173,32862	161,06177	492327	63	Convolutional
venus8KS	174,04248	170,37268	492573	418	Convolutional
Astrounat_Hol_v2	10,57271	10,57407	102247	1635	Fourier
dice2_Hol_v2	10,50599	10,48453	102247	1636	Fourier
Skull_Holo_v2	10,45382	10,57504	102104	1636	Fourier

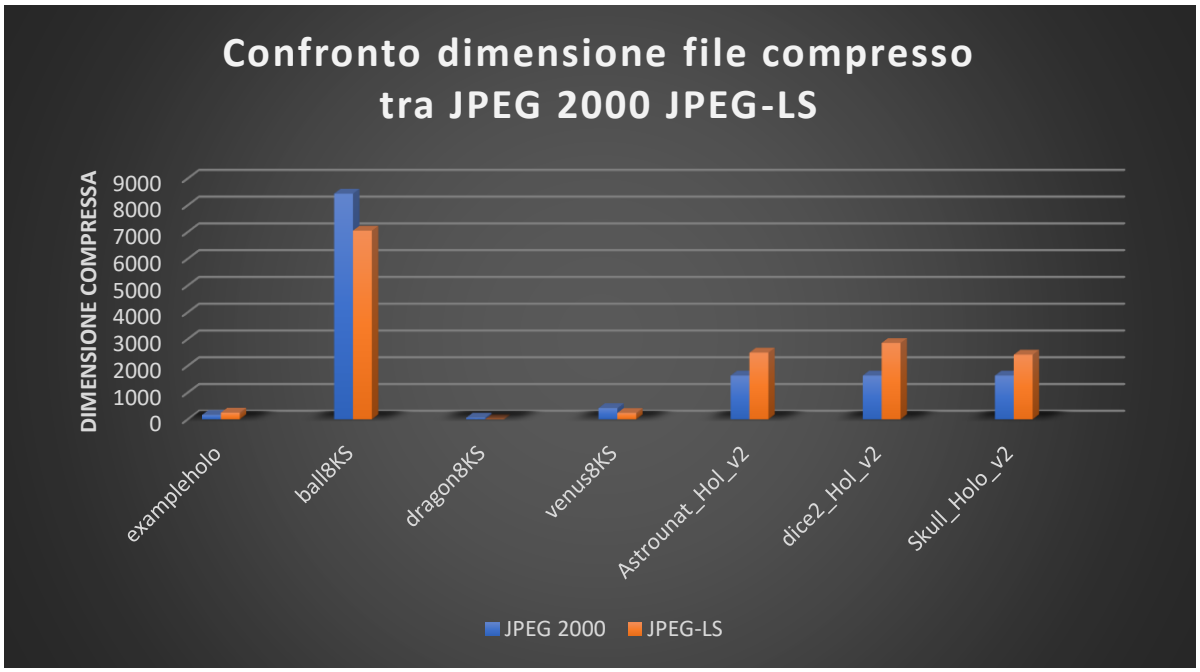
La tabella mostra l'estratto dei testing utilizzando algoritmo di compressione JPEG 2000. Il tempo medio di compressione e decompressione è espresso in secondi, mentre la dimensione dell'ologramma e del file compresso in Kb.

6.4 Test JPEG-LS

Nome	Tempo medio compressione	Tempo medio decompressione	Dimensione iniziale	Dimensione compressa	Metodo
exampleholo	1,52746	1,53605	1176	249	Convolutional
ball8KS	177,4742	175,0987	493532	7047	Convolutional
dragon8KS	172,4958	166,5227	492327	8	Convolutional
venus8KS	171,7337	174,5941	492573	238	Convolutional
Astrounat_Hol_v2	10,80401	10,51204	102247	2494	Fourier
dice2_Hol_v2	10,89844	10,56752	102247	2852	Fourier
Skull_Holo_v2	10,89544	10,42354	102104	2416	Fourier

La tabella mostra l'estratto dei testing utilizzando algoritmo di compressione JPEG-LS. Il tempo medio di compressione e decompressione è espresso in secondi, mentre la dimensione dell'ologramma e del file compresso in Kb.

Dal primo confronto tra i file compressati tramite i due algoritmi lossless possiamo evincere che JPEG 2000 si comporta meglio in termini di compressione per il dataset EmergImg-HoloGrail, mentre per il dataset Interfere-II risulta migliore JPEG-LS.

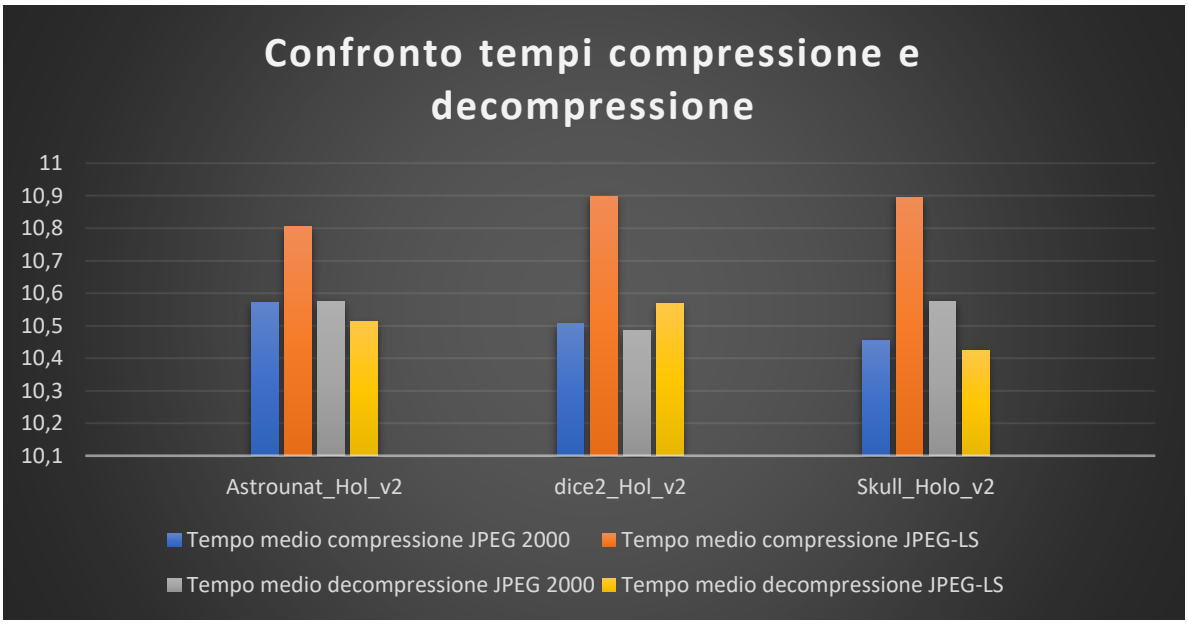


In particolare, nel dataset Interfere-II si nota come il file compresso con JPEG-LS è minore rispetto a JPEG 2000, il contrario succede per il dataset EmergImg-HoloGrail.

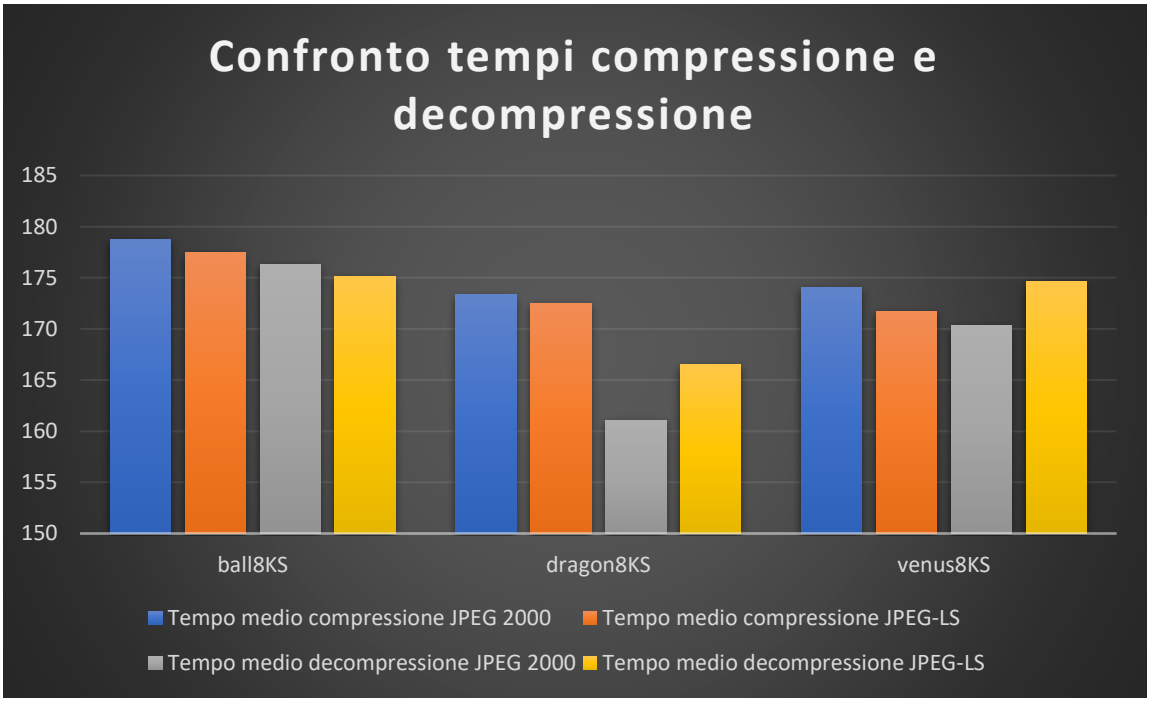
In questa tabella mostra i guadagni in termini di Kb delle dimensioni del file compresso rispetto JPEG 2000.

Nome	JPEG 2000	JPEG-LS	dimensione JPEG 2000 rispetto JPEG-LS
exampleholo	164	249	-85
ball8KS	8431	7047	1384
dragon8KS	63	8	55
venus8KS	418	238	180
Astrounat_Hol_v2	1635	2494	-859
dice2_Hol_v2	1636	2852	-1216
Skull_Holo_v2	1636	2416	-780

Il confronto dei tempi di compressione è schematizzato nei seguenti grafico.



Come si poteva ipotizzare JPEG-LS per gli ologrammi del dataset EmergImg-HoloGrail ha tempi maggiori (circa 2- 3 decimi di secondo), che non sono molti. Da notare come i tempi medi di decompressione per JPEG 2000 risultano identici o leggermente maggiori rispetto ai tempi di compressione.



Per il dataset Interfere-II abbiamo che algoritmo JPEG-LS si comporta in modo migliore in termini di tempo (circa qualche secondo).

Conclusioni

In questo documento viene presentato Hologram Compression Lossless un metodo per la compressione di ologrammi lossless, attraverso la trasformata intera di Fresnel sia la forma convolutiva e sia la forma di Fourier e si è utilizzato due algoritmi di compressione lossless JPEG 2000 e JPEG-LS.

Dai test effettuati che si è osservato in media, i guadagni per gli ologrammi generati dal computer dal repository di ologrammi e dal database Interfere-II sono maggiori di quelli acquisiti otticamente dal database EmergImg-HoloGrail v2. Una possibile ragione di questa differenza potrebbe essere dovuta a fenomeni come il rumore speckle (si ottiene quando un'onda coerente viene fatta passare attraverso un mezzo disordinato) o leggere aberrazioni ottiche. Inoltre, si è notato che in media i tempi di compressione di JPEG-LS sono minori rispetto a JPEG 2000. Da tenere in considerazione JPEG2000 è molto più complicato in termini di *complessità computazionale* rispetto allo standard JPEG. Come è stato spiegato già nel *capitolo 3.6.7 in confronto a JPEG-LS che ha questi vantaggi capitolo 3.10*.

Un altro fattore che si è notato nel capitolo 5, l'ologramma compresso con l'algoritmo di compressione JPEG 2000 risulta avere una qualità migliore rispetto a JPEG-LS che risulta ad avere un'immagine abbastanza rumorosa. Quindi avendo effettuato tutte queste considerazioni si può concludere che non possiamo definire nel nostro caso specifico quale algoritmo di compressione risulta essere il migliore da utilizzare, ma dipende sempre dall'ologramma in input.

Come ricerca futura è possibile includere la Trasformata di Fresnel intera su un insieme più ampio di ologrammi (come i dati per il microscopio olografia), combinandolo con trasformazioni e codec più avanzati e generalizzando lo schema di sollevamento ad altre trasformate canoniche lineari.

Riferimenti

[1] M. Liebling, T. Blu, and M. Unser, “Fresnelets: New multiresolution wavelet bases for digital holography,” Trans. Img. Proc., vol. 12, no. 1, pp. 29–43, Jan. 2003.

[2] Le Thanh Bang, Zulfiqar Ali, Pham Duc Quang, Jae-Hyeung Park, and Nam Kim, “Compression of digital hologram for three-dimensional object using wavelet-bandelets transform,” Opt. Express, vol. 19, no. 9, pp. 8019–8031, Apr 2011.

[3] David Blinder, Tim Bruylants, Heidi Ottevaere, Adrian Munteanu, and Peter Schelkens, “JPEG 2000-based compression of fringe patterns for digital holographic microscopy,” Optical Engineering, vol. 53, no. 12, pp. 123102, 2014.

[4] Yafei Xing, Mounir Kaaniche, Batrice Pesquet-Popescu, and Frdric Dufaux, “Adaptive nonseparable vector lifting scheme for digital holographic data compression,” Appl. Opt., vol. 54, no. 1, pp. A98–A109, Jan 2015.

[5] Tobias Birnbaum, David Blinder, Colas Schretter, and Peter Schelkens, “Compressing
macroscopic near-field digital holograms with wave atoms,” in Imaging and Applied Optics 2018. 2018, p. DW2F.5, Optical Society of America.

[6] J. P. Peixeiro, C. Brites, J. Ascenso, and F. Pereira, “Holographic data coding: Benchmarking
and extending hevc with adapted transforms,” IEEE Transactions on Multimedia, vol. 20, no. 2, pp. 282–297, Feb 2018.

[7] Anas El Rhammad, Patrick Gioia, Antonin Gilles, Arco Cagnazzo, and Beatrice Pesquet-Popescu, “Color digital hologram compression based on matching pursuit,” Applied Optics, vol. 57, no. 17, pp. 4930–4942, June 2018.

[8] D. Blinder, C. Schretter, H. Ottevaere, A. Munteanu, and P. Schelkens, “Unitary transforms using time-frequency warping for digital holograms of deep scenes,” IEEE Transactions on Computational Imaging, vol. 4, no. 2, pp. 206–218, June 2018.

[9] Emmanouil Darakis and John J. Soraghan, “Reconstruction domain compression of phase-shifting digital holograms,” *Appl. Opt.*, vol. 46, no. 3, pp. 351–356, Jan 2007.

[10] Marco V. Bernardo, Pedro Fernandes, Angelo Arrifano, Marc Antonini, Elsa Fonseca, Paulo T. Fiadeiro, Antnio M.G. Pinheiro, and Manuela Pereira, “Holographic representation:
Hologram plane vs. object plane,” *Signal Processing: Image Communication*, vol. 68, pp. 193–206, Oct. 2018.

[11] http://www.crit.rai.it/eletel/2006-3/63-1.pdf

[12]https://cris.vub.be/ws/portalfiles/portal/47151470/Blinder_IEEE_DCC_2019_Accepted_Author_Manuscript_.pdf

[13] https://en.wikipedia.org/wiki/Fresnel_diffraction

[14] Tomoyoshi Shimobaba, Jiantong Weng, Takahiro Sakurai, Naohisa Okada, Takashi

Nishitsuji, Naoki Takada, Atsushi Shiraki, Nobuyuki Masuda, and Tomoyoshi Ito, “Computational wave optics library for c++: Cwo++ library,” Computer Physics Communications, vol. 183, no. 5, pp. 1124 – 1138, 2012.

[15] Wim Sweldens, “The lifting scheme: A custom-design construction of biorthogonal wavelets,” Applied and Computational Harmonic Analysis, vol. 3, no. 2, pp. 186 – 200, 1996.

[16] R. Geiger, Y. Yokotani, G. Schuller, and J. Herre, “Improved integer transforms using multi-dimensional lifting,” in 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, May 2004, vol. 2, pp. ii–1005.

[17]https://cloudinary.com/blog/the_great_jpeg_2000_debate_analyzing_the_pros_and_cons_to_widespread_adoption

[18] D. Blinder, A. Ahar, A. Symeonidou, Y. Xing, T. Bruylants, C. Schretter, B. Pesquet-Popescu, F. Dufaux, A. Munteanu e P. Schelkens, “Database ad accesso aperto per le convalide sperimentali di motori a compressione olografica”, in 2015 Settimo workshop internazionale sulla qualità dell'esperienza multimediale (QoMEX), maggio 2015, pp. 1–6, http://erc-interfere.eu/downloads.html.

[19] Emergimg-holograil holographic database,” http://emergimg.di.ubi.pt/downloads.html.