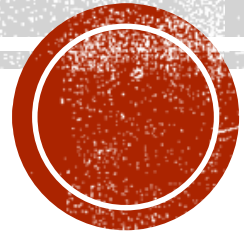
COMPRESSIONE DI IMMAGINI TRAMITE LA DCT

Paolo Marconi 807172

Simone Monti 807994

Gianluca Puleri 807064



OBIETTIVI

Prima parte

- Implementazione della DCT2 in un ambiente open source
- Studio degli effetti della compressione (jpeg sulle immagini in toni di grigio)
- Confrontare i tempi di esecuzione della DCT2 (utilizzando una libreria versione FAST)

Seconda parte

- Scrivere un software che:
  - Permetta all'utente di:
    - Caricare un file immagine *.bmp*
    - Scegliere l'ampiezza della finestra in cui effettuare la DCT2
    - Scegliere la soglia di taglio delle frequenze
  - Creare una nuova immagine suddividendola in blocchi F x F e applicando i seguenti passi ad ogni blocco:
    - Applicare DCT2
    - Eliminare le frequenze c_{kl} con $k + l \geq d$
    - Applicare la DCT2 inversa all'array c così modificato: ff = IDCT2(c)
    - Arrotondare ff all'intero più vicino
  - Confrontare le immagini

SOFTWARE UTILIZZATI

Per lo sviluppo del progetto sono state utilizzate le seguenti risorse:

- Eclipse, ambiente di sviluppo
- jTransforms, libreria utilizzata per le *trasformate di Fourier*

eclipse + Libreria jTransforms

Eclipse può essere utilizzato per la produzione di software di vario genere, noi lo abbiamo utilizzato come un IDE per il linguaggio Java.

JTransforms è la prima libreria open source, multithreaded FFT scritta in puro Java. Attualmente, quattro tipi di trasformate sono disponibili: Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Discrete Sine Transform (DST) and Discrete Hartley Transform (DHT).

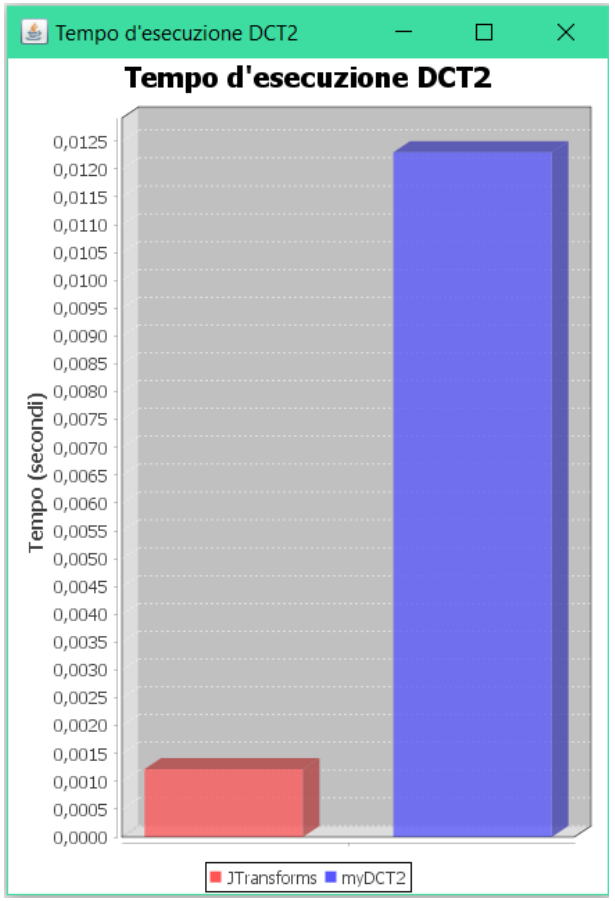
Github: https://github.com/wendykierp/JTransforms

IMPLEMENTAZIONE DELLA DCT2

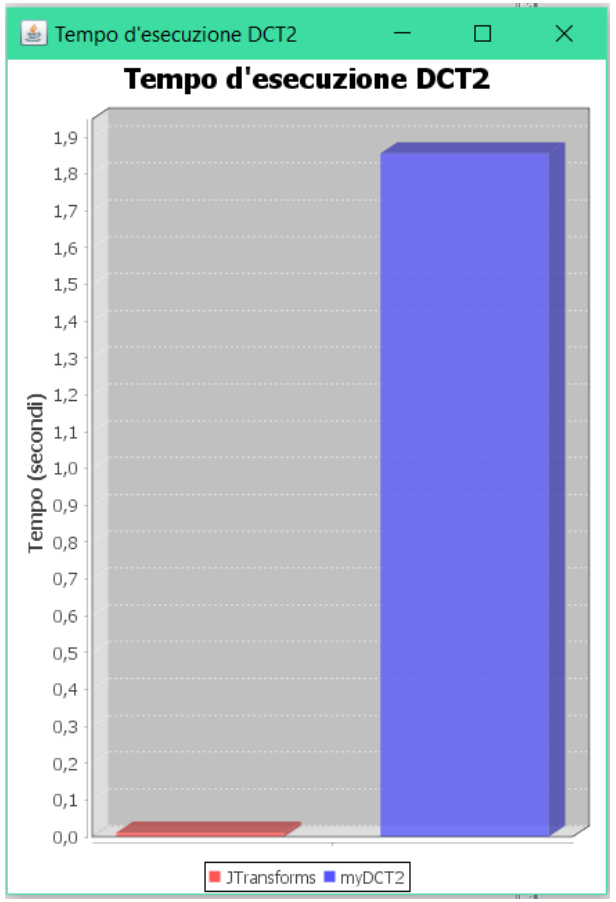
```java
public class myDCT2 {

    public myDCT2() {}

    public void DCT1(double[] f) {
        double[] app = new double [f.length];
        double alphak;
        for(int k = 0; k < f.length; k++) {
            if(k == 0)
                alphak = Math.sqrt(1.0) / Math.sqrt(f.length);
            else
                alphak = Math.sqrt(2.0) / Math.sqrt(f.length);
            double sum = 0;
            for (int i = 0; i < f.length; i++)
                sum += f[i] * Math.cos(k * Math.PI * ((2 * i + 1) / (2.0 * f.length)));
            app[k] = alphak * sum;
        }

        for(int i = 0; i < f.length; i++)
            f[i] = app[i];
    }

    public void DCT2(double[][] f) {
        double[][] app = new double [f[0].length][f.length];

        // transpose the matrix
        for(int i = 0; i < f[0].length; i++)
            for(int j = 0; j < f.length; j++)
                app[i][j] = f[j][i];

        // calculate monodimensional DCT on "column"
        for(int i = 0; i < f[0].length; i++)
            DCT1(app[i]);

        // retranspose the matrix
        for(int i = 0; i < f.length; i++)
            for(int j = 0; j < f[0].length; j++)
                f[i][j] = app[j][i];

        // calculate monodimensional DCT on rows
        for(int i = 0; i < f.length; i++)
            DCT1(f[i]);
    }
}
```

TEMPI DI ESECUZIONE

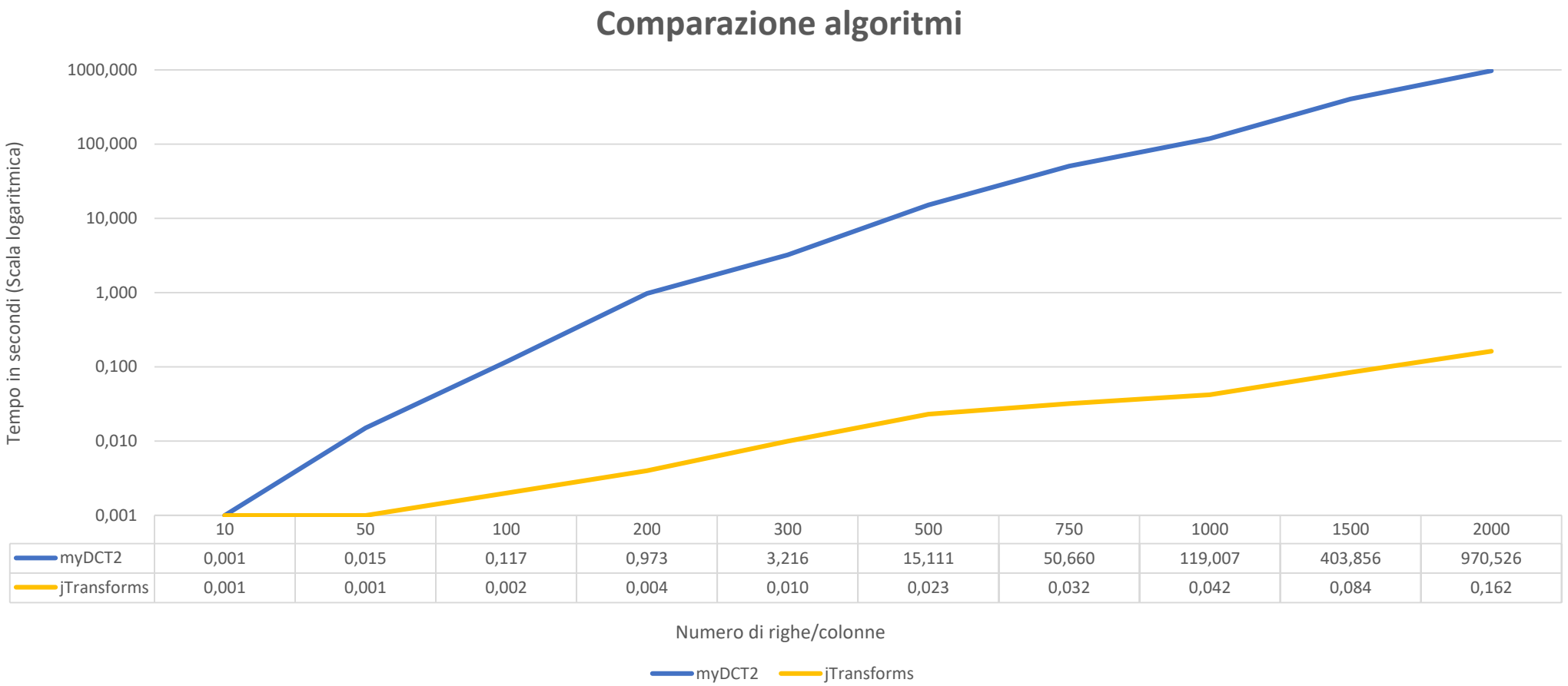
- **N = 40**



- **N = 250**



TEMPI DI ESECUZIONE - GRAFICO



	10	50	100	200	300	500	750	1000	1500	2000
myDCT2	0,001	0,015	0,117	0,973	3,216	15,111	50,660	119,007	403,856	970,526
jTransforms	0,001	0,001	0,002	0,004	0,010	0,023	0,032	0,042	0,084	0,162

INTERFACCIA DEL SOFTWARE

- Semplice
- Permette all'utente di inserire:
  - Un'immagine *.bmp* direttamente dal filesystem
  - Un intero che indica l'ampiezza della finestra in cui si effettuerà la DCT2 – campo *F*
  - Un intero compreso tra *0* e $(2F-2)$ che indica la soglia di taglio delle frequenze – campo d

SUDDIVISIONE DELL'IMMAGINE IN BLOCCHI F X F

```
// number of image's block
int nImgWidth = this.getWidth() / this.getF();
int nImgHeight = this.getHeight() / this.getF();

// divide the whole image in nImgWidth * nImgHeight images
double[][][] img = new double[nImgWidth * nImgHeight][this.getF()][this.getF()];
for (int k = 0; k < nImgWidth; k++) {
    for (int w = 0; w < nImgHeight; w++) {
        for (int i = 0; i < this.getF(); i++) {
            for (int j = 0; j < this.getF(); j++) {
                img[k * nImgHeight + w][i][j] =
                        this.image[k * this.getF() + i][w * this.getF() + j];
            }
        }
    }
}
```

CICLO EFFETTUATO PER OGNI BLOCCO DELL'IMMAGINE

```
DoubleDCT_2D idct2 = new DoubleDCT_2D(this.getF(), this.getF());

// for every block
for (int n = 0; n < nImgWidth * nImgHeight; n++) {
    // apply DCT
    idct2.forward(img[n], true);

    // delete frequencies where k + l >= d
    for (int k = 0; k < this.getF(); k++) {
        for (int l = 0; l < this.getF(); l++) {
            if (k + l >= this.getD())
                img[n][k][l] = 0;
        }
    }

    // apply reverse DCT
    idct2.inverse(img[n], true);
    for (int i = 0; i < this.getF(); i++) {
        for (int j = 0; j < this.getF(); j++) {
            // round and set consistent data
            Math.round(img[n][i][j]);
            if (img[n][i][j] > 255.0)
                img[n][i][j] = 255;
            else if (img[n][i][j] < 0.0) {
                img[n][i][j] = 0;
            }
        }
    }
}
```

RICOMPOSIZIONE DELL'IMMAGINE

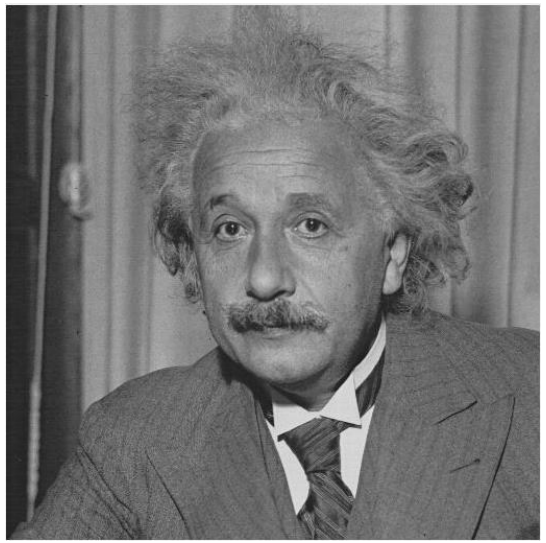
```
        // recompose the whole image
        for (int k = 0; k < nImgWidth; k++) {
            for (int w = 0; w < nImgHeight; w++) {
                for (int i = 0; i < this.getF(); i++) {
                    for (int j = 0; j < this.getF(); j++) {
                        this.image[k * this.getF() + i][w * this.getF() + j] =
                                img[k * nImgHeight + w][i][j];
                    }
                }
            }
        }

        return this.pixelsToImage(this.image);
    }
```

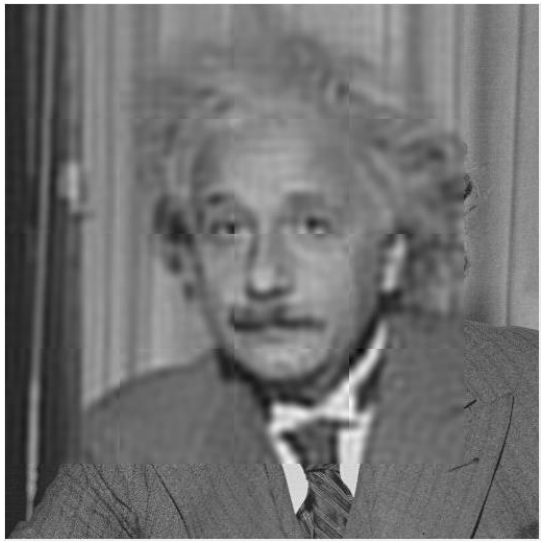
FUNZIONI DI SUPPORTO

```
public double[][] imageToPixels(BufferedImage image) {
    Raster raster = image.getData();
    this.width = raster.getWidth();
    this.height = raster.getHeight();
    double[][] pixels = new double[this.getWidth()][this.getHeight()];
    for (int i = 0; i < this.getWidth(); i++) {
        for (int j = 0; j < this.getHeight(); j++) {
            pixels[i][j] = raster.getSample(i, j, 0);
        }
    }

    return pixels;
}

public BufferedImage pixelsToImage(double[][] pixels) {
    BufferedImage image = new BufferedImage(this.getWidth(), this.getHeight(),
            BufferedImage.TYPE_BYTE_GRAY);
    WritableRaster wr = image.getRaster();
    for (int i = 0; i < this.getWidth(); i++) {
        for (int j = 0; j < this.getHeight(); j++) {
            wr.setSample(i, j, 0, pixels[i][j]);
        }
    }
    return image;
}
```

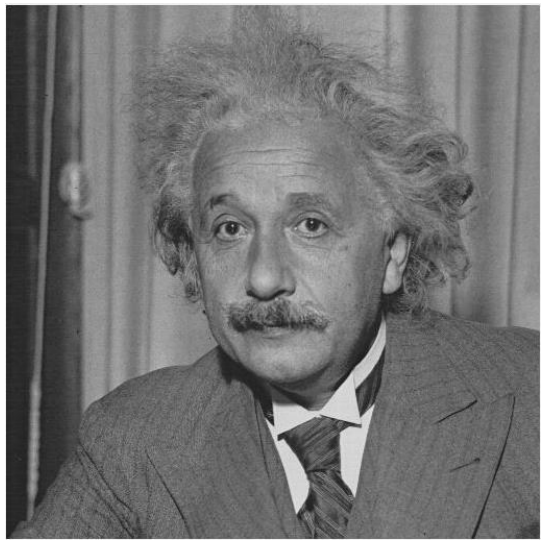
IMMAGINI A CONFRONTO

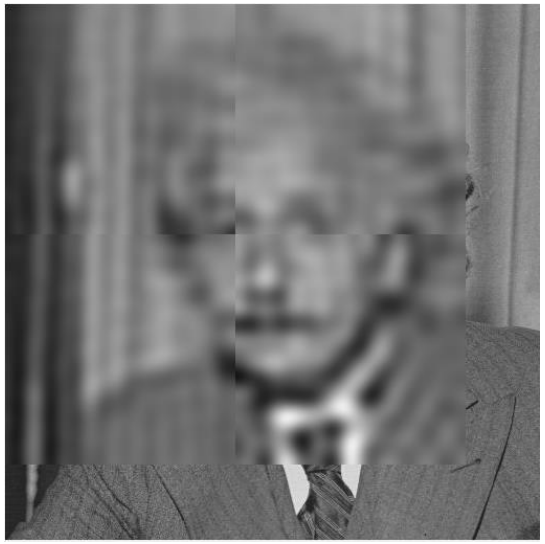
- F = 110, d = 20



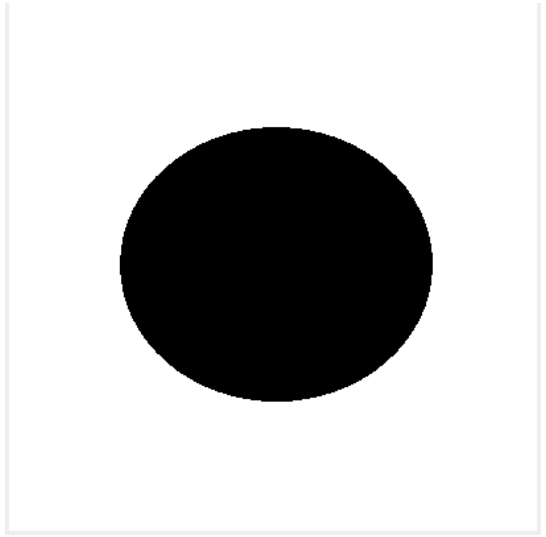


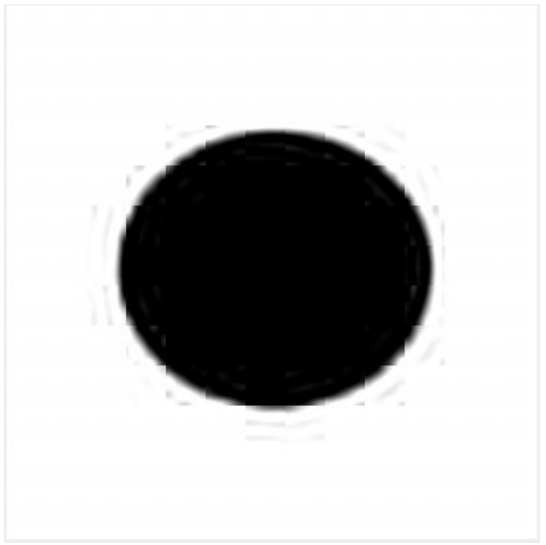
- F = 220, d = 20



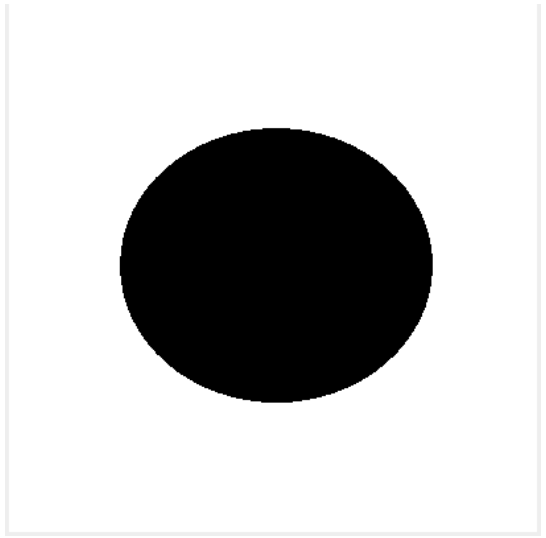


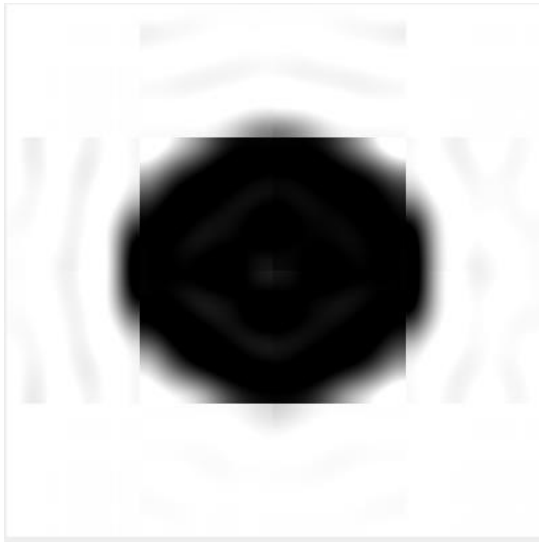
- F = 30, d = 6





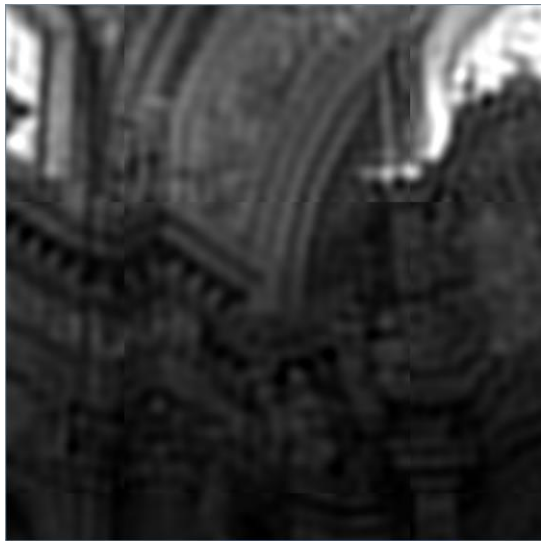
- F = 100, d = 6



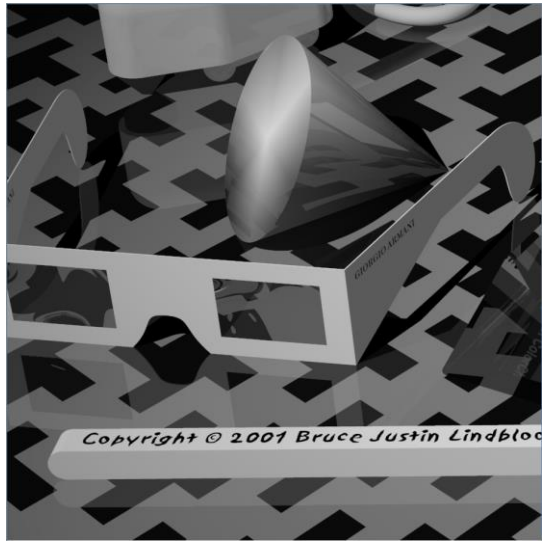


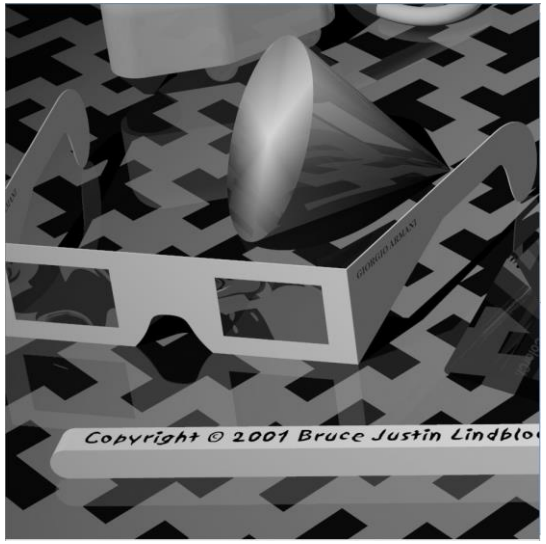
- F = 500, d = 120

- F = 500, d = 40

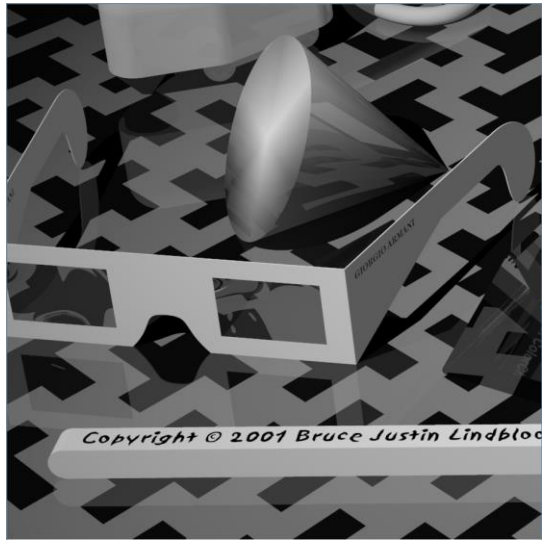


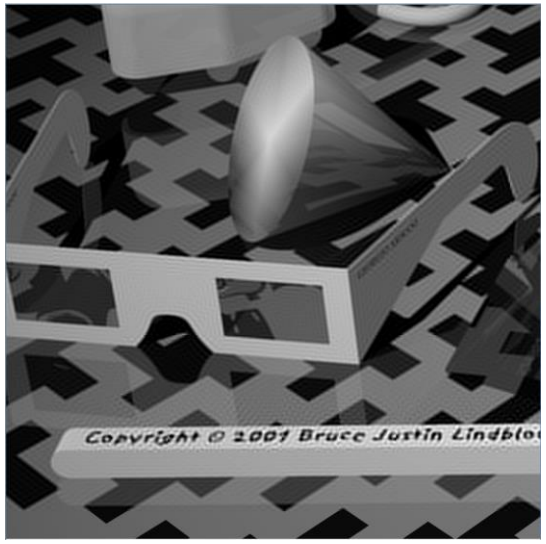
- F = 100, d = 120





- F = 100, d = 30





CONCLUSIONI

- È stato possibile constatare che già da N molto piccoli (es. 40) il tempo di esecuzione dell'algoritmo DCT2 da noi implementato è di un ordine di grandezza maggiore a quello della libreria, e pertanto per N elevati diventa pressoché inutilizzabile.

- Dagli esperimenti eseguiti su diverse foto è emerso che all'aumentare di F o al diminuire di d, l'immagine perde di qualità. Per valori di d compresi nella scala dello 0-20% rispetto al valore di F si riescono a vedere le varie finestre in cui l'immagine è stata suddivisa. Invece, per valori di d oltre il 100% rispetto al valore di F, l'immagine sembra quasi non perdere di qualità.