UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Metodi del Calcolo Scientifico - Progetto 2

Compressione di immagini tramite la DCT

Alberici Federico - 808058
Bettini Ivo Junior - 806878
Cocca Umberto - 807191
Traversa Silvia - 816435

Anno Accademico 2019 - 2020

Indice

Introduzione

In questa relazione vengono presentate e discusse le modalita di implementazione della DCT (dall'inglese Discrete Cosine Transform), ovvero la più diffusa funzione che provvede alla compressione spaziale.
Nella prima parte viene confrontata la versione "Nativa" della DCT con alcune varianti conosciute, studiandone il costo computazionale.
Nella seconda parte viene documentato un semplice tool per applicare su immagini in toni di grigio, tramite un approccio di compressione tipo jpeg (senza utilizzare una matrice di quantizzazione), la funzione DCT2 implemetata.

Analisi DCT

1.1 Discrete Cosine Transform

Una DCT esprime una sequenza finita di punti in termini di una somma di funzioni coseno oscillanti a diverse frequenze. Ad oggi è una delle tecniche di trasformazione piu utlizzate nella Teoria dei segnali e nella compressione dei dati, in particolare nei media digitali (audio, video, radio ecc..).
In queste applicazioni infatti la maggior parte delle informazioni significative tendono ad essere concentrate in poche componenti a bassa frequenza della DCT. Questo permette di comprimere a piacere il dato scartando le componenti ad alta frequenza (compressione lossy).

1.2 DCT e IDCT

La DCT-II è probabilmente la forma più utilizzata, infatti viene indicata come "la DCT".

$$C_k = \alpha_k \sum_{i=0}^{N-1} V_i \cos\left[\frac{\pi\,(2i+1)\,k}{2N}\right] \quad i = 0, \ldots, N-1 \quad e \quad \alpha_k = \begin{cases} 1/\sqrt{N}, & \text{se } k = 0 \\ \sqrt{2/N}, & \text{se } 1 \leq k \leq N-1 \end{cases}$$

La sua inversa è la DCT-III e per questo viene indicata come "l'inversa della DCT" o "IDCT".

$$V_i = \sum_{k=0}^{N-1} \alpha_k C_k \cos\left[\frac{\pi\,(2i+1)\,k}{2N}\right] \quad k = 0, \ldots, N{-}1 \quad e \quad \alpha_k = \begin{cases} 1/\sqrt{N}, & \text{se } k = 0 \\ \sqrt{2/N}, & \text{se } 1 \leq k \leq N-1 \end{cases}$$

Entrambe le funzioni effettuano N somme per calcolare la k-esima componente di un vettore di N componenti, determinando un costo computazionale $O(N^2)$.

Implementazione

Per l'implementazione è stato utilizzato C++, sfruttando la libreria open-source Eigen (`https://eigen.tuxfamily.org/`) per semplificare la gestione dei dati.

```cpp
void DCT2::DCT(Eigen::VectorXd &_v)
{

  const Eigen::VectorXd _v_copy = _v;
  const int N = _v.size();
  double ak = 1.0 / sqrt(N);
  double ck = 0;

  for (int k = 0; k < N; k++)
  {
    ck = 0;
    for (int i = 0; i < N; i++)
    {
      ck += cos((2.0 * i + 1.0) * k * M_PI / (2.0 * N)) * _v_copy(i);
    }
    _v(k) = ak * ck;
    if (k == 0)
    {
      ak = sqrt(2.0) / sqrt(N);
    }
  }
}
```

Listato 1.1: Funzione di calcolo DCT

```
void DCT2::IDCT(Eigen::VectorXd &_c)
{
    const Eigen::VectorXd _c_copy = _c;
    const int N = _c.size();
    double ak = 0;
    double vi = 0;

    for (int i = 0; i < N; i++)
    {
        vi = 0;
        ak = 1.0 / sqrt(N);
        for (int k = 0; k < N; k++)
        {
            vi += cos((2.0 * i + 1.0) * k * M_PI / (2.0 * N)) * _c_copy(k) * ak;
            if (k == 0)
            {
                ak = sqrt(2.0) / sqrt(N);
            }
        }
        _c(i) = vi;
    }
}
```

Listato 1.2: Funzione di calcolo IDCT

1.3 DCT2 e IDCT2

La DCT2 è una trasformazione a due dimensioni, ottenuta semplicemente applicando la DCT mono-dimensionale ad una dimensione, seguita da un'altra applicazione all'altra dimensione.

La definizione della DCT bi-dimensionale per una matrice A m x n in input è:

$$C_{kl} = \alpha_k \alpha_l \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij} \cos\left[\frac{\pi\,(2i+1)\,k}{2m}\right] \cos\left[\frac{\pi\,(2j+1)\,l}{2n}\right],$$

$$\mathit{con} \quad 0 \le k \le m-1, \quad 0 \le l \le n-1,$$

$$\alpha_k = \begin{cases} 1/\sqrt{m}, & \text{se } i = 0 \\ \sqrt{2/m}, & \text{se } 1 \le i \le m-1 \end{cases} \quad e \quad \alpha_l = \begin{cases} 1/\sqrt{n}, & \text{se } j = 0 \\ \sqrt{2/n}, & \text{se } 1 \le j \le n-1 \end{cases}$$

L'inversa di tale trasformazione è la IDCT2, ottenuta applicando IDCT alle due dimensioni.

Implementazione

Anche in questo caso Eigen è utilizzato per mantenere la struttura dati tramite un oggetto *Eigen::MatrixXd*. L'implementazione non sfrutta direttamente la definizione ma computa la DCT2/IDCT2 prima sulle righe e poi sulle colonne della matrice in input. Inoltre essendo l'elaborazione di ogni vettore indipendente dagli altri è possibile parallelizzarne la computazione.

```
Eigen::MatrixXd DCT2::DCT2_mt(Eigen::MatrixXd &_m)
{
  Eigen::MatrixXd out = _m;

// DCT su righe
#pragma omp parallel for
  for (int i = 0; i < out.rows(); i++)
  {
    Eigen::VectorXd row = out.row(i);
    DCT(row);
    out.row(i) = row;
  }

// DCT su colonne
#pragma omp parallel for
  for (int i = 0; i < out.cols(); i++)
  {
    Eigen::VectorXd col = out.col(i);
    DCT(col);
    out.col(i) = col;
  }

  return out;
}
```

Listato 1.3: Funzione di calcolo DCT2

```cpp
Eigen::MatrixXd DCT2::IDCT2_mt(Eigen::MatrixXd &_m)
{
  Eigen::MatrixXd out = _m;

// IDCT su righe
#pragma omp parallel for
  for (int i = 0; i < out.rows(); i++)
  {
    Eigen::VectorXd row = out.row(i);
    IDCT(row);
    out.row(i) = row;
  }

// IDCT su colonne
#pragma omp parallel for
  for (int i = 0; i < out.cols(); i++)
  {
    Eigen::VectorXd col = out.col(i);
    IDCT(col);
    out.col(i) = col;
  }

  return out;
}
```

Listato 1.4: Funzione di calcolo IDCT2

1.4 Varianti DCT

Esistono diverse varianti della DCT che riducono la complessita a $O(NlogN)$. Tali metodi sono conosciuti come *fast DCT* o *FCT* in quanto appunto migliorano notevolmente il costo computazionale.
Di seguito vengono citate due delle più comuni.

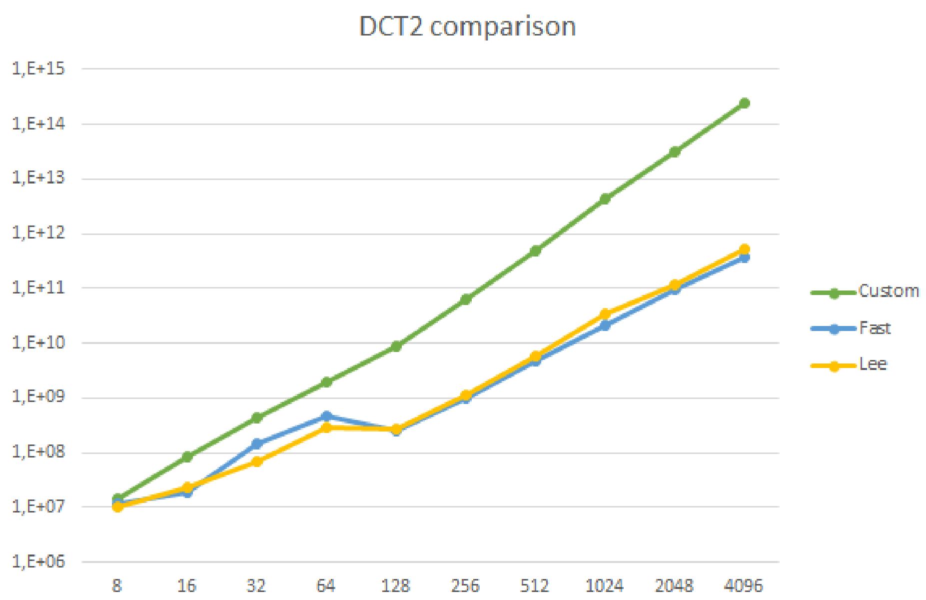
Fast DCT di Lee

Descritta da Byeong Gi Lee [1] nel 1984 è uno degli algoritmi fast DCT per 2^m punti più comune. Utilizza una struttura ricorsiva dove la trasformazione DCT è decomposta in una parte pari e una dispari. Queste parti sono a loro volta decomposte nello stesso modo finchè non sono abbastanza piccole (N=2) da essere calcolate tramite valutazione diretta [3].

Fast DCT FFT

Invece di applicare direttamente la formula DCT (o scomporla come mostrato da Lee) è possibile fattorizzare la computazione in modo simile alla *fast Fourier transform* (FFT). Gli algoritmi basati su sull'algoritmo di Cooley-Tukey [2] sono i piu comuni, ma qualunque altro algoritmo FFT è applicabile.

1.5 Confronto



Test con immagini

2.1 Tool di testing

2.2 Risultati

Bibliography

[1] Byeong Lee. “A new algorithm to compute the discrete cosine Transform”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.6 (1984), pp. 1243–1245.

[2] James W. Cooley and John W. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90 (1965), pp. 297–301. ISSN: 00255718, 10886842. URL: `http://www.jstor.org/stable/2003354`.

[3] KRISTER LAGERSTRÖM. “Design and Implementation of an MPEG-1 Layer III Audio Decoder”. In: 2001. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.3056&rep=rep1&type=pdf#page=34`.