Threshold-Free Cluster Enhancement

Luigi Giugliano[1], Marco Mecchia[1]

[1]Università degli studi di Salerno

9 maggio 2016

Overview

Overview

TFCE

Threshold-Free Cluster Enhancemen

Molte tecniche di image enhancing usano informazioni spaziali per aumentare l'autenticità di estese aree di segnale. La motivazione nell'utilizzo di voxel vicini spazialmente per aumentare la significatività delle regioni del segnale è da ritrovarsi nel fatto che le regioni del segnale sono più estese del rumore e quindi trovare zone aumenta la possibilità che esse siano segnale vero e proprio e non rumore.

Overview

Codice

Andremo ora a spiegare il codice prodotto per il plugin TFCE.

I file principali che compongono il plugin sono:

- Tfce.cpp
- Utilities.cpp

Tfce è il core del plugin, dove avviene il calcolo degli score.

Utilities invece contiene tutti le funzioni di supporto per l'esecuzione del plugin stesso.

L' unica funzione che viene esposta dal file **Tfce.h** è:

```
#ifndef TFCE_H
    #define TFCE_H
    #include <float.h>
     float * tfce_score(float * map, int dim_x, int
         dim_y, int dim_z, float E, float H, float dh);
#endif //TFCE_H
```

Le funzioni che espone **Utilities.h** sono:

```
#ifndef UTILITIES_H
    #define UTILITIES_H

    #include <float.h>
    #include <stdio.h>

    void findMinMax(float *map, int n, float *min,
        float *max, float * range);

    int confront(float a, float b, char operation);

    int * getBinaryVector(float * map, int n, int
        (*confront)(float, float), float value, int *
            numOfElementsMatching);
```

```
float * fromBinaryToRealVector(float * map, int n,
    int * binaryVector);

float * fill0(int n);

void apply_function(float * vector, int n, float (*
    operation) (float a, float b), float argument);

int linearIndexFromCoordinate(int x, int y, int z,
    int max_x, int max_y);

void coordinatesFromLinearIndex(int index, int
    max_x, int max_y, int * x, int * y, int * z);

float * copyAndConvertIntVector(int * vector, int
    n);

#endif //UTILITIES_H
```

Andremo ora a vedere l'implementazione della funzione **tfice_score**:

```
findMinMax(map, n, &minData, &maxData, &rangeData);
precision = rangeData/dh;
if (precision > 200) {
    increment = rangeData/200;
} else{
    increment = rangeData/precision;
}
steps = ceil((maxData - minData) / (increment));
#pragma omp parallel for
for (i = 0; i < steps; i++) {
     computeTfceIteration(minData + i*increment, map,
         n, dim_x, dim_y, dim_z, E, H, dh, toReturn);
}
return toReturn;
```

Funzione **computeTfceIteration**:

```
int * indexMatchingData = getBinaryVector(map, n,
    moreThan, h, &numOfElementsMatching);
clustered_map = find_clusters_3D(indexMatchingData,
    dim_x, dim_y, dim_z, n, &num_clusters);
extent_map = new int[n];
for (j = 0; j < n; ++j){
    extent_map[j] = 0;
}
delete [] indexMatchingData;
for (i = 1; i <= num_clusters; ++i) {
    numOfElementsMatching = 0;
    for (j = 0; j < n; ++j){
        if(clustered_map[j] == i){
            numOfElementsMatching++; }}
    for (j = 0; j < n; ++j) {
       if(clustered_map[j] == i)
           extent_map[j] = numOfElementsMatching; }}
```

```
clustered_map_float =
    copyAndConvertIntVector(extent_map, n);
apply_function(clustered_map_float, n, elevate, E);
apply_function(clustered_map_float, n, multiply,
    pow(h, H));
apply_function(clustered_map_float, n, multiply, dh);
for (i = 0; i < n; ++i) {
#pragma omp atomic
    toReturn[i] += (clustered_map_float[i]);
}
delete[] clustered_map_float;
delete[] clustered_map;
delete[] extent_map;
```

Funzione **getBinaryVector**:

```
int * getBinaryVector(float * map, int n, int
    (*confront)(float, float), float value, int *
    numOfElementsMatching){
    int * binaryVector = new int [n];
    (*numOfElementsMatching) = 0;
    int i;
    for (i = 0; i < n; ++i) {
        if(confront(map[i],value)){
            binaryVector[i] = 1;
            (*numOfElementsMatching)++;
        }
        else
            binaryVector[i] = 0;
        }
    return binaryVector;
}
```

Funzione **find_cluster_3D**:

```
int * find_clusters_3D(int * binaryVector, int dim_x,
    int dim_y, int dim_z, int n, int * num_clusters)
```

questo metodo preso in input una **mappa binaria in 3D ma linearizzata**, le sue tre dimensioni, il numero totale voxel della mappa e il puntatore ad un intero che indica il numero attuale di cluster trovati.

Cercando i cluster all'interno della mappa 3D utilizzando la **26-connectivity**

Restituisce un'altra mappa in cui al posto degli uno viene sostituito l'identificativo del cluster.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 2 & 2 \\ 1 & 0 & 0 & 2 & 2 \end{bmatrix}$$

In questo piccolo esempio in 2D viene mostrato il funzionamento della nostra funzione.

E' stato utilizzata la specifica OpenMP per rendere il calcolo degli score più veloce.

OpenMP (Open Multiprocessing) è un API multipiattaforma per la creazione di applicazioni parallele su sistemi a memoria condivisa.

Il comando:
#pragma omp parallel for
viene utilizzato per rendere un for parallelo.

Il comando:
#pragma omp atomic
invece viene utilizzato per rendere un istruzione atomica.

Abbiamo deciso di utilizzare, OMP perché l'effort per utilizzarlo è praticamente nullo, e le prestazioni sono ottime.

Inoltre essendo che l'implementazione dei *Thread* in *C* cambia tra Windows e Linux, si sarebbe reso necessario modificare il codice per renderlo funzionante su entrambe le piattaforme.