ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Dipartimento Informatica – Scienza e Ingegneria

Corso di Laurea in Ingegneria e Scienze Informatiche

IMPLEMENTAZIONE IN LINGUAGGIO C++ IN VERSIONE OTTIMIZZATA DEL TOOL RECONSTRUCTION AND VISUALIZATION FROM A SINGLE PROJECTION (REVISP)

Elaborato in

Programmazione

Relatore

Prof. Antonella Carbonaro

Presentata da

Sofia Belloni

Correlatori

Ing. Filippo Piccinini

Prof. Giovanni Martinelli

Anno Accademico 2020 / 2021

“*Misura ciò che è misurabile,*

e rendi misurabile ciò che non lo è.”

Galileo Galilei

PAROLE CHIAVE

Biomedical Imaging

Microscopy

Image Processing

Open Source Code

Oncology

ABSTRACT (English Version)

Volume is one of the most important features for the characterization of a tumor on a macroscopic scale. It is often used to assess the effectiveness of care treatments, thus making its correct assessment a crucial issue for patient care. Similarly, the volume is a fundamental characteristic even on a microscopic scale, but in the literature there are very few methods proposed to estimate the volume of tumor spheroids, 3D tumor models of about 1 mm in diameter, created in vitro and widely used in pre-clinical studies to test the effects of drugs and radiotherapy treatments. Among the methods proposed to estimate the volume of tumor spheroids, the most used is known as *Reconstruction and Visualization from a Single Projection* (ReViSP). It is an automatic method designed to reconstruct the 3D surface and estimate the volume of multicellular spheroids using a simple 2D projection, which for example could be an image acquired with a simple optical widefield microscope.

ReViSP software is distributed as an open source tool with code developed in MATLAB (The MathWorks, Inc., MA, USA). However, C++ code is more versatile and, since it is licence-free, it suites better with the policy of code distributed in the form of Open Source. In this work, using the MATLAB Coder, OpenCV and several other toolboxes and libraries, we created an optimised version of ReViSP in C++ language. The code is distributed as an open source tool and can be integrated in several C++ frameworks used in oncology for the estimation of radiomics feature of tumors.

ABSTRACT (versione Italiano)

Il volume è una delle feature più importanti per la caratterizzazione di un tumore su scala macroscopica. Viene spesso utilizzato per valutare l'efficacia dei trattamenti assistenziali, rendendo così la sua corretta valutazione una questione cruciale per la cura del paziente. Allo stesso modo, il volume è una caratteristica fondamentale anche su una scala microscopica ma, ad esempio in letteratura, esistono pochissimi metodi proposti per stimare il volume di sferoidi tumorali, modelli tumorali 3D di circa 1 mm di diametro, creati in vitro e ampiamente utilizzati in studi pre-clinici per testare gli effetti dei farmaci e trattamenti radioterapici. Tra i metodi proposti per stimare il volume degli sferoidi tumorali, il più utilizzato è conosciuto con il nome di *Reconstruction and Visualization from a Single Projection* (ReViSP). E' un metodo automatico concepito per ricostruire la superficie 3D e stimare il volume di sferoidi multicellulari utilizzando una semplice proiezione 2D, che ad esempio potrebbe essere un'immagine acquisita con un semplice microscopio ottico a campo largo.

Il software ReViSP è distribuito come strumento open source con codice sviluppato in MATLAB. Tuttavia, il codice C++ risulta essere più versatile e, non essendo collegato a licenze di utilizzo, meglio si adatta alla politica di codice distribuito in forma di Open Source. In questo lavoro, utilizzando il MATLAB Coder, la libreria OpenCV e molti altri toolbox e librerie, abbiamo creato una versione ottimizzata di ReViSP in linguaggio C++. Il codice è distribuito come strumento open source e può essere integrato in diversi framework C++ utilizzati in oncologia per la stima di feature radiomiche dei tumori.

INDICE

Introduzione

Le colture cellulari tridimensionali (3D), rappresentano un modello ampiamente utilizzato in studi di oncologia preclinica. In particolare, organoidi (aggregati 3D con caratteristiche più simili a quelle dei tumori in vivo rispetto alle colture bidimensionali) e sferoidi multicellulari (modelli simili agli organoidi ma di forma tendenzialmente sferica) sono utilizzati nei laboratori biologici fin dagli anni ’70 come modello per lo studio degli effetti di farmaci e trattamenti antitumorali, collocandosi in un livello intermedio tra colture monostrato 2D e modelli animale.

Gli organoidi vengono generalmente acquistati da particolari ditte o prodotti con procedure particolari aggregando cellule di diversi tipi. Invece, gli sferoidi generalmente sono facilmente creati in laboratorio partendo da singole cellule tumorali. Esistono ad oggi vari sistemi utilizzati per la generazione degli sferoidi. Tra i più comuni troviamo le piastre a basso attrito (*low attachment plates),* i sistemi a goccia sospesa (*hanging drop plates*), e i biorettori antigravitazionali. Una volta generati, gli sferoidi vengono collocati in tipiche piastre multi-pozzetto (*multi-well plates*), uno per ogni pozzetto, e vengono osservati con microscopi standard a campo largo in grado di acquisire immagini 2D sfruttando o la classica luce ottica se le cellule non sono marcate, oppure fluorescenza in caso di utilizzo di apposite sonde. È quindi procedura standard ottenere per ogni sferoide una singola immagine ed osservare, a seguito di segmentazione (procedura conosciuta anche con il termine “contornazione” che indica l’identificazione dei bordi dell’oggetto di interesse – *foreground* – a discapito dello sfondo – *background*), alcune feature morfologiche (dimensioni, compattezza, frastagliatura dei bordi) per dedurre cambiamenti biologici delle singole cellule componenti.

Tuttavia, tra le varie feature, quando si parla di tumori, quella storicamente più utilizzata per la valutazione degli effetti di trattamenti radio-farmacologici, è il volume. Generalmente, un calo di volume della massa tumorale è associato ad un effetto positivo della cura, mentre un aumento del volume tumorale è generalmente un primo campanello di allarme a riguardo di necessità di variazioni di farmaci o trattamenti. Tra i metodi proposti per stimare il volume degli sferoidi tumorali, il più utilizzato è conosciuto con il nome di *Reconstruction and Visualization from a Single Projection* (ReViSP). Si tratta di un metodo automatico concepito per ricostruire la superficie 3D e stimare il volume di sferoidi multicellulari utilizzando una

semplice proiezione 2D, che ad esempio potrebbe essere un'immagine acquisita con un semplice microscopio ottico a campo largo. L'articolo scientifico che ha introdotto ReViSP nella comunità è stato pubblicato dalla rivista Computer Methods and Programs in Biomedicine nel 2015 e la sua citazione completa è la seguente: "*F. Piccinini, A. Tesei, C. Arienti, A. Bevilacqua, Cancer multicellular spheroids: Volume assessment from a single 2D projection. Computer Methods and Programs in Biomedicine, 118(2):95–106, 2015. DOI: 10.1016/j.cmpb.2014.12.003"* **[1]**. Ad oggi, questo articolo è stato citato 39 volte.

ReViSP è un software distribuito come strumento open source con codice sviluppato in MATLAB. Al giorno d'oggi, MATLAB risulta infatti essere uno degli ambienti informatici di maggior diffusione soprattutto tra Ingegneri e Ricercatori operanti nel campo sanitario. In particolare, MATLAB rappresenta un potente linguaggio di programmazione ottimizzato per il calcolo matematico basato su matrici multidimensionali e, proprio per queste sue caratteristiche, è il linguaggio più comune nel mondo biomedicale. Tuttavia, risulta essere un ambiente *closed source*: la necessità di dover disporre o acquistare una licenza di utilizzo rappresenta un limite per la diffusione e l'utilizzo nella comunità del codice sviluppato. Dall'altro lato, C++ è un codice *open source* che non richiede nessuna licenza né per lo sviluppo né per la distribuzione, e per questo motivo risulta essere ampiamente diffuso nella comunità, sebbene richieda maggiori competenze di programmazione in fase di sviluppo codice. Trattandosi però di linguaggi sintatticamente e concettualmente molto diversi, la traduzione di un software da MATLAB e C++ è tutt'altro che immediata. Tuttavia, gli sviluppatori della MathWorks (The MathWorks, Inc., Massachusetts, USA) hanno recentemente rilasciato un toolbox dal nome MATLAB Coder contenente varie funzioni per semplificare la traduzione del codice tra questi due linguaggi di programmazione.

Una versione di ReViSP in linguaggio C++ risulterebbe quindi più versatile e potrebbe essere integrata in diversi framework C++ utilizzati in oncologia per la stima di feature radiomiche dei tumori. Sebbene infatti nel corso del tempo siano state proposte diverse formule per stimare il volume di un tumore, generalmente supponendo come modello una sfera o un ellissoide, nessuna di queste sembra fornire risultati accurati. Gli sferoidi, sebbene di forma tendenzialmente più sferica rispetto agli organoidi e i tumori in generale, sono tendenzialmente caratterizzati da una morfologia 3D comunque complessa che, se semplificata ad una semplice sfera o in un modello ellissoide, porta ad una rappresentazione nella maggior parte dei casi irrealistica e poco utile per le valutazioni cliniche. Inoltre è stato

appurato che alcuni metodi comunemente utilizzati per stimare il volume di uno sferoide sono generalmente caratterizzati da un errore medio molto elevato, superiore al 10% e gli autori hanno dimostrato che ReViSP è l'unico metodo che ha fornito risultati soddisfacenti anche in caso di protuberanze dello sferoide: "*F. Piccinini, A. Tesei, A. Bevilacqua, Single-image based methods used for non-invasive volume estimation of cancer spheroids: a practical assessing approach based on entry-level equipment. Computer Methods and Programs in Biomedicine, 135:51-60, 2016. DOI: 10.1016/j.cmpb.2016.07.024*" **[2]**.

Questo testo è stato redatto durante un progetto di Tesi triennale in Ingegneria e Scienze Informatiche, svolto in collaborazione tra il *Dipartimento Informatica – Scienza e Ingegneria* dell'*Università di Bologna* con sede a Cesena e l'*IRCCS Istituto Romagnolo per lo Studio dei Tumori (IRST) "Dino Amadori"* (via Piero Maroncelli 40, 47014 Meldola, FC, Italia) e si inserisce nel contesto di un progetto multidisciplinare di collaborazione tra il gruppo di ricerca *Microscopy & Artificial intelligence* (MiAi, IRST), coordinato dall'Ing. Filippo Piccinini, ed il gruppo di ricerca *Data Science For Health* (DS4H, Università di Bologna), coordinato dalla Prof.ssa Antonella Carbonaro. L'obbiettivo di questa tesi è stato quello di realizzare una versione ottimizzata del software ReViSP in linguaggio C++, al fine di poterlo facilmente integrare in framework per la stima di feature radiomiche. L'algoritmo originale non risulta ad oggi ottimizzato per gestire immagini in diversi formati e in batch. Partendo quindi da un'attenta analisi dei singoli blocchi di cui è composto il programma, abbiamo proceduto a creare una traduzione in C++ che tenga conto delle caratteristiche intrinseche di questo linguaggio che consente una gestione più puntuale della memoria allocata per le operazioni, sebbene, a differenza di MATLAB, non sia pensato specificatamente per il calcolo numerico. Inoltre, nel periodo di preparazione ai lavori, nella fase in cui si approfondiva l'utilizzo del MATLAB Coder per capire quali funzioni era possibile sfruttare per convertire codice MATLAB in codice C++, abbiamo pubblicato con la casa editrice italiana YouCanPrint un libro sia in lingua Inglese sia in lingua Italiana, dal titolo: "*F. Piccinini, S. Belloni, English and Italian Book, Guide for Dummies: from MATLAB to C++ through the MATLAB Coder, Guida per Dummies: da MATLAB a C++ attraverso il MATLAB Coder, Edition 2021*" **[3]**. Infine, per divulgare maggiormente il codice C++ di ReViSP, abbiamo distribuito il tutto in forma di open source attraverso *FigShare*, un portale pubblico che non richiede nessuna identificazione.

Questo elaborato è composto dai capitoli di seguito indicati.

1. Capitolo 1: descrizione dei Metodi e Materiali utilizzati dal punto di vista oncologico, con particolare attenzione agli sferoidi, ai microscopi ottici a campo largo ed alla possibilità di acquisire immagini 2D con altri strumenti per diagnosi.
2. Capitolo 2: descrizione specifica dei Materiali e Metodi utilizzati lato informatico, con digressione sulle versioni utilizzate di MATLAB, MATLAB Coder, Microsoft Visual Studio e libreria OpenCV.
3. Capitolo 3: descrizione del software ReViSP approfondendo il metodo proposto per il calcolo del volume e relativa implementazione in C++.
4. Capitolo 4: descrizione degli esperimenti svolti per validare il codice sviluppato confrontando l'implementazione originale in MATLAB con quella proposta in C++.
5. Capitolo 5: digressione sui problemi principali incontrati durante la conversione del codice e l'ottimizzazione dello stesso in linguaggio C++.
6. Capitolo 6: conclusioni del lavoro svolto ed enfatizzazione dei principali sviluppi futuri.

CAP.1: Materiali e Metodi – Lato Oncologico

1.1 Sferoidi multicellulari

Gli sferoidi sono dei sistemi multicellulari tridimensionali (3D) che permettono di ricreare *in vitro* alcuni degli aspetti isto-morfologici, funzionali e micro-ambientali dei tessuti tumorali *in vivo*, collocandosi ad un livello intermedio tra colture monostrato 2D e modelli animali. E' importante specificare che la comunità scientifica si è ormai largamente espressa sul fatto che le colture in 2D, infatti, sono in grado di imitare le condizioni di un tessuto fisiologico solo in parte, mentre le cellule *in vivo* hanno la capacità di interagire in una rete tridimensionale più facilmente simulata attraverso modelli *in vitro* 3D. Proprio per questo, i risultati generati dalle colture monostrato sono ad oggi spesso poco considerati per valutare la futura efficacia clinica di farmaci e trattamenti e sono utilizzate principalmente per esperimenti iniziali sulla tossicità delle procedure e componenti utilizzati. Le colture cellulari 3D rappresentano invece un ottimo modello sperimentale su cui poter testare gli effetti dei farmaci e dei trattamenti terapeutici in quanto, possedendo una struttura 3D, sono in grado di imitare più fedelmente le condizioni delle cellule tumorali reali. Tra le colture cellulari 3D identifichiamo gli organoidi, aggregati 3D formati da cellule con diverso fenotipo e quindi per questo con caratteristiche che richiamano quelle reali del tumore in fase di analisi, e sferoidi multicellulari, caratterizzati da uno o più aggregati di forma tendenzialmente sferica.

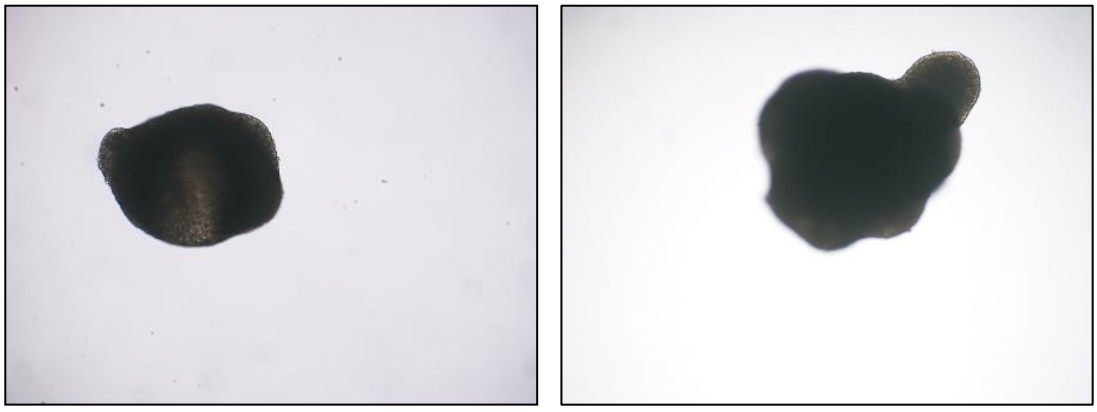


Fig. 1.1: Immagine di sferoidi (**a**) senza protuberanze e (**b**) con protuberanze, acquisite con microscopio ottico in luce visibile.

La prima creazione documentata di colture cellulari tridimensionali in laboratorio a partire da cellule tumorali risale agli inizi degli anni ’70 ad opera di R. M. Sutherland e collaboratori, ed ha avviato l’utilizzo di tali tecniche in campo oncologico per l’esecuzione di test in fase preclinica. Ad oggi esistono vari sistemi utilizzati per la generazione degli sferoidi. Tra i più comuni troviamo le piastre a basso attrito (*low attachment plates)*, i sistemi a goccia sospesa (*hanging drop plates*), e i biorettori antigravitazionali. Una volta generati, gli sferoidi vengono collocati in tipiche piastre multi-pozzetto (*multi-well plates*) e vengono osservati con microscopi standard a campo largo in grado di acquisire immagini 2D a fuoco sfruttando generalmente la classica luce ottica (o la fluorescenza in caso di utilizzo di sonde). È quindi procedura standard ottenere un’immagine di ogni sferoide e osservare alcune caratteristiche morfologiche (dimensioni, compattezza, frastagliatura dei bordi) per individuare cambiamenti morfologici e biologici delle singole cellule componenti. In particolare, risulta di fondamentale importanza una precisa valutazione del volume del tumore che consente di esaminare l’efficacia del trattamento oggetto di valutazione.

1.2 Microscopi ottici a campo largo ed altri sistemi di visualizzazione

Il microscopio è uno strumento che consente di ingrandire l'immagine di organismi, o altri oggetti di piccole dimensioni con l'aiuto di lenti, e permette di studiarne i dettagli mediante un’osservazione diretta ad occhio nudo, oppure indiretta tramite la fotografia **[4]**. Il microscopio ottico è un tipo di microscopio che sfrutta la luce con lunghezza d'onda dal vicino infrarosso all'ultravioletto. L’utilizzo di luce nell’intervallo di lunghezze d’onda 400nm–700nm, ovvero lo spettro della luce visibile, ha due principali vantaggi: permette al campione di non essere danneggiato e di essere osservato direttamente dall’occhio umano, motivo per cui la microscopia ottica con luce visibile è tuttora una scelta comune in campo biologico.

I microscopi ottici a campo largo sono considerati i microscopi classici di un laboratorio biologico e si distinguono dalla macroclasse dei microscopi ottici confocali per la capacità di acquisire (visualizzare) campi di vista x-y in un unico istante, senza bisogno di scannerizzare il campione come avviene ad esempio nei microscopi confocali. I microscopi ottici a campo

largo sono caratterizzati generalmente da due sistemi di lenti: oculare e obiettivo, rispettivamente quello in cui viene appoggiato l'occhio e quello vicino all'oggetto da osservare, dall'altra estremità del tubo.

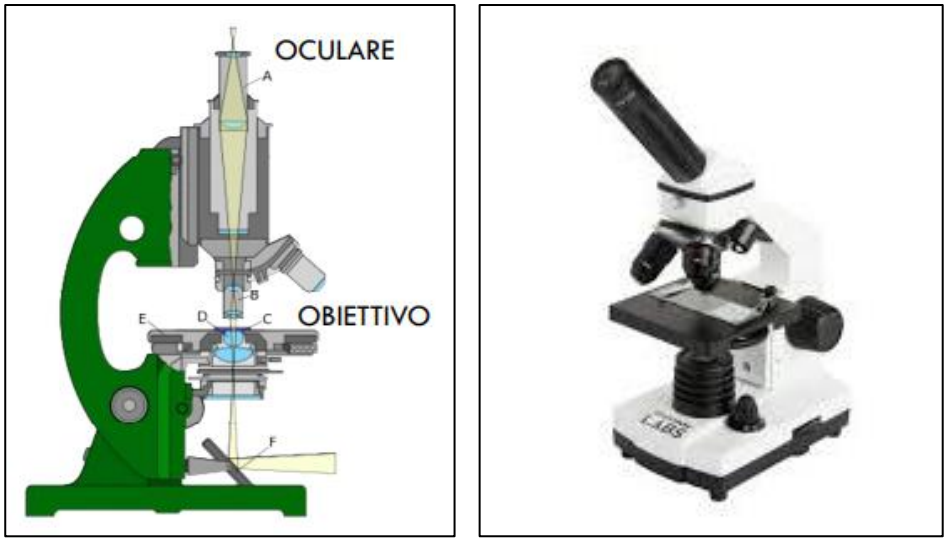


Fig. 1.2: Microscopio ottico diritto con oculare in alto e obiettivo posto sopra al tavolo porta oggetto.

Per consentire analisi automatizzate, al microscopio viene poi generalmente collegata una telecamera digitale in grado di acquisire immagini digitali. Queste sono tipicamente processate con software specifici per diversi tipi di analisi.

ReViSP nasce per l'analisi di immagini 2D di sferoidi ottenute con microscopi a campo largo. Tuttavia, se l'oggetto da analizzare è caratterizzato da una simmetria circolare locale (nei prossimi capitoli spiegheremo meglio questo concetto), nulla vieta di procedere alla stima dell'oggetto applicando l'algoritmo alla base di ReViSP utilizzando ad esempio immagini 2D conosciute con il nome di *maximum intensity projectioni* ottenute con un microscopio confocale o a foglietto di luce, oppure con sistemi TAC, PET, SPECT o di risonanza magnetica. Per onestà, è doveroso però far notare che l'algoritmo di ReViSP attualmente è stato scientificamente validato solo utilizzando immagini di sferoidi acquisiti con microscopi a campo largo.

CAP. 2: Materiali e Metodi – Lato Informatico

Tutte le analisi svolte in questo lavoro sono state condotte utilizzando un consumer laptop di marca HP, Windows 64-bit, processore AMD Ryzen 5, con scheda video Radeon Vega 8. I principali software utilizzati sono stati:

- MATLAB 2020b
- MATLAB Coder™ versione 5.1
- Visual Studio 2019

Nei seguenti sotto-capitoli vengono descritte in dettaglio le caratteristiche di questi programmi/tool e verranno illustrate alcune nozioni di base sulla gestione delle immagini in C++.

2.1 MATLAB 2020b

MATLAB, abbreviazione di *MATrix LABoratory*, è un ambiente di programmazione commerciale ottimizzato per applicazioni scientifiche, elaborazioni numeriche e per la simulazione di sistemi dinamici. Con il termine MATLAB si intende anche direttamente l'omonimo linguaggio di programmazione creato dalla MathWorks (The MathWorks, Inc., Massachusetts, USA). Oggi MATLAB è uno degli ambienti di sviluppo scientifici di maggior diffusione nel mondo biomedicale e viene utilizzato in molti corsi universitari e aziende del campo ingegneristico perché grazie ai tanti toolbox (i.e. collezione di programmi e funzioni che permettono la soluzione di problemi mirati a campi specifici) risulta molto versatile e semplice da utilizzare. Il 16 settembre 2020 è stata ufficialmente rilasciata la release 2020b, utilizzata in questo testo.

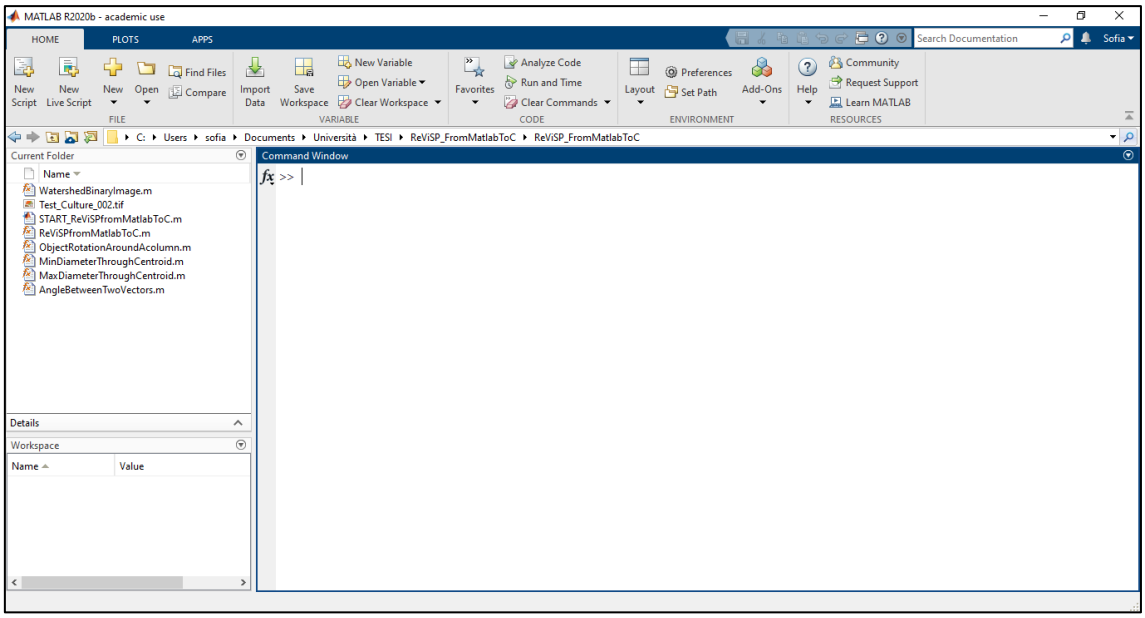


Fig. 2.1: MATLAB R2020b.

2.2 MATLAB Coder™ 5.1

Il MATLAB Coder permette di generare codice C e C++ a partire da codice MATLAB che può poi essere facilmente integrato in progetti C/C++ come codice sorgente, librerie statiche ed eseguibili. Il MATLAB Coder supporta buona parte del linguaggio MATLAB ed una vasta gamma di toolbox. Tuttavia, non tutte le funzioni ed i comandi MATLAB sono supportati dal MATLAB Coder (ad esempio non è supportato il comando "*clear*" per pulire il workspace) e spesso è opportuna una intensa fase di debugging del codice MATLAB prima di procedere alla traduzione. È inoltre possibile includere il codice generato come funzione MEX da usare nello stesso MATLAB. In questo lavoro è stata utilizzata la versione 5.1 del MATLAB Coder.

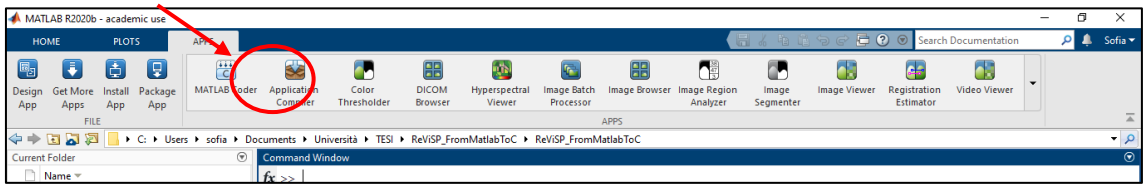


Fig. 2.2: L'applicazione MATLAB Coder si trova nel tab "*APPS*" di MATLAB.

A riguardo della licenza del codice generato utilizzando il MATLAB Coder, mostriamo quanto citato nel sito della MathWorks (printscreen salvato in data 16/06/2021):

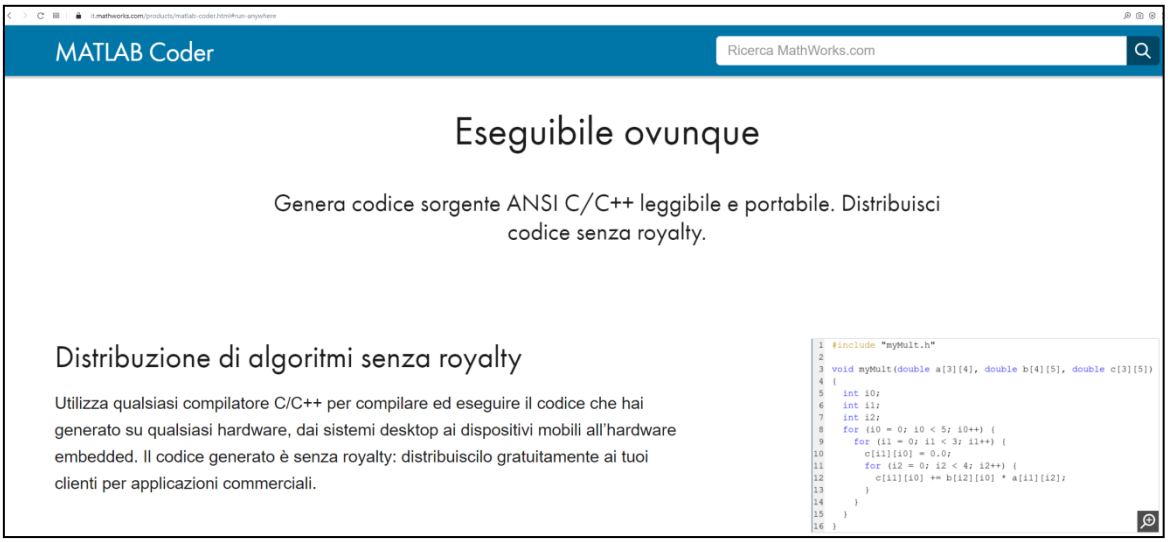


Fig. 2.3: *https://it.mathworks.com/products/matlab-coder.html* (copyright MathWorks).

Come si legge: "*Il codice generato è senza royalty: distribuiscilo gratuitamente ai tuoi clienti per applicazioni commerciali*". Tuttavia, è importante notare che la maggior parte dei file generati dal MATLAB Coder è protetta dalla seguente licenza che lascia libero l'utilizzo a scopo di didattica e ricerca, ma limita l'utilizzo a scopo commerciale.

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
* government, commercial, or other organizational use.
*/
```

2.3 Visual Studio 2019

Visual Studio (Microsoft, Washington, USA) è uno dei principali ambienti di sviluppo integrato (IDE) per i linguaggi C e C++. L'ultima versione è la 2019, disponibile per Windows in tre edizioni: Community, Professional ed Enterprise. In questo lavoro è stata

utilizzata la versione Community, gratuitamente utilizzabile da tutta la comunità senza nessuna necessità di licenza.

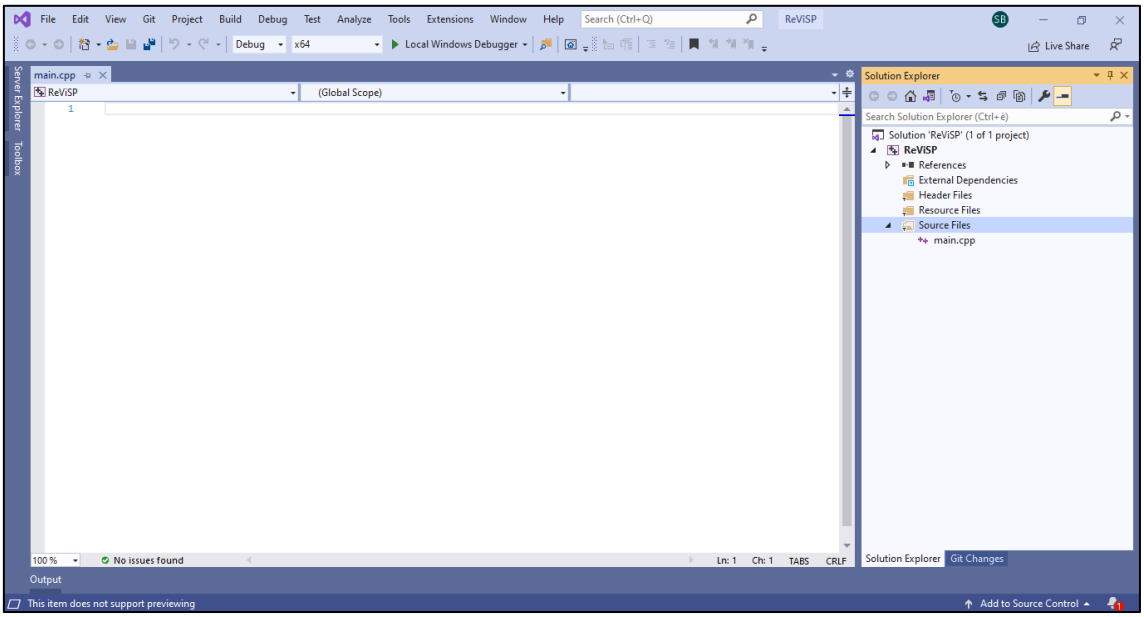


Fig. 2.4: Visual Studio 2019.

2.4 OpenCV 4.5.1

La libreria OpenCV è una delle librerie open-source più utilizzate dagli sviluppatori di codice C/C++ per la gestione di immagini digitali. Un'immagine digitale è la rappresentazione numerica di un'immagine bidimensionale, rappresentata tramite una matrice di pixel. Esistono tre diversi modi di rappresentazione di un'immagine:

- Immagine binaria, in cui ogni pixel è rappresentato da 1 bit, il quale può assumere il valore 0, corrispondente al colore nero, oppure 1, corrispondente al colore bianco.
- Immagine in scala di grigi (*grayscale*), in cui ogni elemento della matrice è un valore intero compreso tra 0 e 255, e rappresenta uno dei 256 livelli di grigio.
- Immagine a colori RGB, in cui ogni elemento della matrice corrisponde a una terna ordinata di interi (R, G, B) che indicano rispettivamente le intensità di rosso, verde e blu e con valori compresi tra 0 e 255, corrispondenti ai 256 diversi livelli di intensità delle tre sorgenti di colore. In tal caso un'immagine può essere vista come una terna di matrici, dove ognuna rappresenta uno dei colori fondamentali . I colori fondamentali RGB sono

additivi, ossia i contributi individuali di ciascuno sono sommati insieme per produrre il risultato.

Per la gestione delle immagini in C++, in particolar modo per la loro visualizzazione ed il loro salvataggio, si è scelto di utilizzare la libreria OpenCV **[5]**, una libreria open-source impiegata principalmente nella Computer Vision, che permette una manipolazione delle immagini piuttosto semplice, in cui queste vengono trattate come 'matrici di pixel'. A partire dalla versione 4.0 OpenCV non offre più il supporto al linguaggio C, motivo per cui in questo testo è stato scelto C++ come principale linguaggio di riferimento.

Per prima cosa, è necessario installare la libreria di cui, in questo caso, è stata utilizzata la versione 4.5.1. In Windows l'installazione consiste semplicemente nell'esecuzione del file di installazione eseguibile (*.exe*), scaricabile da GitHub. Durante il setup si consiglia di scegliere come directory di installazione il disco C (*"C:\"*). Successivamente si dovranno specificare le directory di include e le librerie in tutti i progetti che richiedono l'utilizzo di funzioni OpenCV. In Visual Studio 2019 questa operazione richiede i seguenti semplici passi:
Step1: Aprire il progetto Visual Studio 2019 al quale si vuole includere la libreria OpenCV ed assicurarsi di averlo configurato per una piattaforma x64.

Step2: Nella schermata "*Project"* → "*Properties"* aprire il tab "*Configuration Properties"* → "*VC++ Directories"*. A questo punto è necessario aggiungere la directory di include "*C:\opencv\build\include"* e la directory contenente la libreria "*C:\opencv\build\x64\vc15\lib".* Al termine si dovrebbe ottenere una schermata analoga a quella riportata in **Fig. 1.5**.

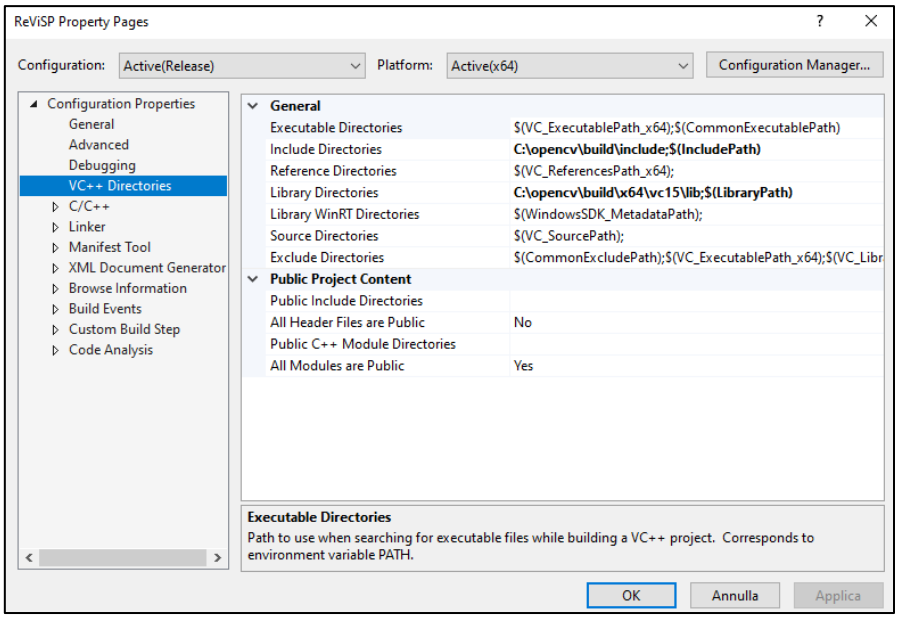


Fig. 2.5: Schermata "*Configuration Properties*" → "*VC++ Directories*" correttamente configurata.

Step3: Nella stessa schermata "*Project*" → "*Properties*" aprire il tab "*Linker*" → "*Input*" ed aggiungere la libreria "*opencv_world451.lib*", presente nella directory "*C:\opencv\build\x64\vc15\lib*". Il nome della libreria potrebbe essere diverso in base alla release scaricata. Al termine si dovrebbe ottenere una schermata analoga a quella riportata in **Fig. 2.6**.

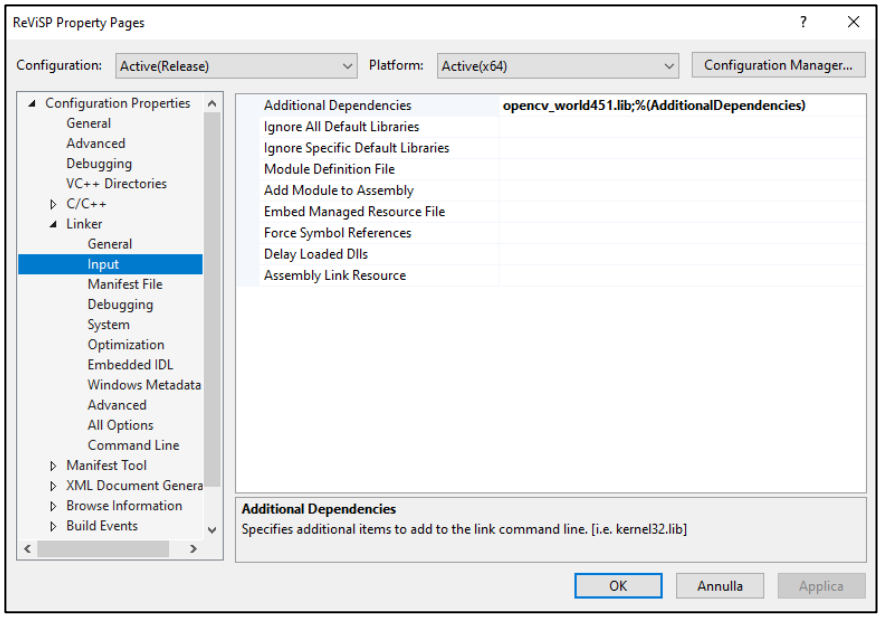


Fig. 2.6: Schermata "*Linker*" → "*Input*" correttamente configurata.

A questo punto, all'interno del progetto, è possibile utilizzare tutte le funzioni e le classi definite nella libreria. In particolare, queste si trovano tutte all'interno del namespace *cv* e, pertanto, per accedere a queste funzionalità dal codice è necessario utilizzare lo specificatore "*cv::"* oppure la direttiva "*using namespace cv;".*

In OpenCV un'immagine è gestita tramite la struttura dati *cv::Mat*, composta essenzialmente da due parti: un'intestazione e un blocco dati.

```
MatAllocator *  allocator;
        int cols;
      uchar * data;
  const uchar *dataend;
 const uchar *datalimit;
 const uchar *datastart;
       int dims;
      int flags;
       int rows;
    MatSize size;
    MatStep step;
    UMatData * u;
```

L'intestazione, elencata sopra in dettaglio, contiene tutte le informazioni associate alla matrice (come la dimensione, il numero di righe e colonne, etc..) ed un puntatore (attributo *data*) al blocco dati, il quale contiene invece tutti i valori dei pixel dell'immagine. Il motivo di questa implementazione è proprio quello di evitare che vengano copiate inutilmente grandi quantità di dati quando le matrici vengono passate come input alle funzioni. Laddove venga richiesta in input un'istanza di *cv::Mat*, viene passata alla funzione solo l'intestazione della matrice, con la conseguenza che il puntatore farà riferimento alla stessa area di memoria della matrice originale. Analogamente, anche gli operatori di copia copieranno solo le intestazioni ed il puntatore alla matrice, non i dati stessi. Nel caso in cui si desideri copiare anche la matrice stessa, OpenCV fornisce le funzioni *cv::Mat::clone()* e *cv::Mat::copyTo()*. Infine è importante evidenziare che con l'interfaccia C++ di OpenCV non è necessario pensare alla gestione della memoria di una variabile di tipo cv::*Mat*: questa verrà infatti automaticamente allocata e rilasciata quando necessario.

La funzione che consente di caricare un'immagine da disco è la seguente:

```
Mat cv::imread (const String &filename, int flags= IMREAD_COLOR)
```

dove il parametro *filename* rappresenta il nome del file, mentre *flags* può assumere valori diversi in base alla modalità di caricamento, tra cui di default IMREAD_COLOR per leggere immagini a colori BGR a tre canali, e IMREAD_GRAYSCALE per leggerle in scala di grigi, ovvero a canale singolo.

Per la visualizzazione dell'immagine è invece necessario innanzitutto creare la finestra all'interno della quale l'immagine verrà visualizzata attraverso la seguente funzione:

```
void cv::namedWindow (const String &name, int flags=WINDOW_AUTOSIZE)
```

dove *name* rappresenta il nome della finestra che viene visualizzato nella parte superiore della stessa, e *flags* indica se la finestra deve ridimensionarsi automaticamente per adattarsi all'immagine che è stata inserita in essa.

A questo punto, per mostrare l'immagine nella finestra precedentemente creata, è possibile chiamare la seguente funzione:

```
void cv::imshow (const String &name, InputArray image)
```

dove *name* è il nome della finestra all'interno della quale si intende disegnare e *image* è l'immagine da mostrare.

La seguente funzione permette infine di salvare l'immagine nel formato specificato:

```
bool cv::imwrite(const String &filename, InputArray  img,
  const std::vector<int> &params = std::vector<int>())
```

dove *filename* rappresenta il nome del file di destinazione, *img* l'immagine(cv::Mat) da salvare e *params* i parametri specifici del formato codificati come coppie (paramId_1, paramValue_1, paramId_2, paramValue_2, ...). Il formato dell'immagine viene scelto in base all'estensione del nome del file.

2.5 Licenza e repository file

La versione ottimizzata di ReViSP in linguaggio C++ realizzata in questo lavoro è disponibile al seguente link:

https://sourceforge.net/p/revisp/files/

sotto la seguente licenza di utilizzo:

The software and all the materials are copyright protected.

CAP. 3: Reconstruction and Visualization from a Single Projection

Come già anticipato, il volume è una delle caratteristiche più rilevanti di un tumore: una sua stima accurata risulta fondamentale per ottenere dati affidabili su cui si possono basare le valutazioni di trattamenti e terapie antitumorali necessari per la cura del paziente. In letteratura, attualmente sono stati proposti pochissimi metodi per stimare il volume di un tumore a partire da un'unica immagine e la maggior parte di essi si basa sull'approssimazione dello sferoide a modelli nella maggior parte dei casi irrealistici, come modelli a sfera o ellissoide, che non permettono di considerare la reale complessa morfologia 3D dello sferoide, e che portano quindi a risultati poco affidabili per eventuali valutazioni cliniche.

Il software *Reconstruction and Visualization from a Single Projection* (*ReViSP*) è un software specifico per la stima del volume di singoli o colture di sferoidi partendo da una proiezione 2D, quindi una singola immagine, che ad esempio potrebbe essere acquisita con un semplice microscopio ottico a campo largo. Il metodo studiato suppone una simmetria sferica locale e stima la superficie tridimensionale esterna tenendo conto anche delle protuberanze, molto frequenti negli sferoidi di diverse linee cellulari. ReViSP è distribuito come strumento open source con codice sviluppato in MATLAB.

Lo scopo di questo lavoro è proprio quello di ottenere una versione ottimizzata in C++ di ReViSP distribuibile come strumento open source in grado di essere integrato in diversi framework C++ utilizzati in oncologia per la stima di feature radiomiche dei tumori. Trattandosi però di linguaggi sintatticamente e concettualmente molto diversi, la traduzione di un software MATLAB in un codice C++ è tutt'altro che immediata. Per tale ragione si è scelto di sfruttare le potenzialità del MATLAB Coder per convertire il codice in maniera più agile, evitando anche il rischio di introdurre potenziali errori durante il processo di conversione. Inoltre, per le future release del software non sarà difficile mantenere implementazioni MATLAB e C++ sincronizzate: una volta compresi i meccanismi necessari per poter convertire il codice di ReViSP, la traduzione di una sua nuova versione potrà essere quasi immediata.

3.1 Schema a blocchi delle funzioni princiipali

La versione originale del tool ReViSP sviluppato in MATLAB è disponibile in forma open-source e comprende un'interfaccia grafica (GUI) che rende il suo utilizzo semplice e user-friendly. Per poter procedere con la conversione del codice è stato necessario, come prima cosa, dividere il cuore dell'algoritmo per la stima del volume dalla componente grafica. Durante questa fase sono state inoltre individuate diverse sezioni in cui era possibile dividere concettualmente l'algoritmo e per il codice relativo ad ognuna di esse sono state evidenziate una o più funzioni. In particolare sono stati individuati i seguenti blocchi:

- Blocco scheletronizzazione: l'immagine passata in input viene convertita in un'immagine binaria forzata a valori *uint8* ma avente solo 0 e 1 come valori reali ed eliminando gli elementi che toccano i bordi per evitare che possano inficiare l'analisi successiva. Al termine si restituisce un'immagine che consente di distinguere i singoli sferoidi grazie ad una numerazione progressiva dei pixel partendo da 1 fino al numero complessivo degli sferoidi stessi. Successivamente, ogni singolo sferoide viene processato per isolare, a partire dall'immagine originale, il singolo oggetto da analizzare andando a riempire eventuali buchi, intesi come pixel di valori 0 che non possono essere raggiunti riempiendo lo sfondo partendo dal bordo dell'immagine. Infine restituisce l'immagine "ritagliata" sul bordo dello sferoide per estrarlo dal contesto e ridurre l'area analizzata.

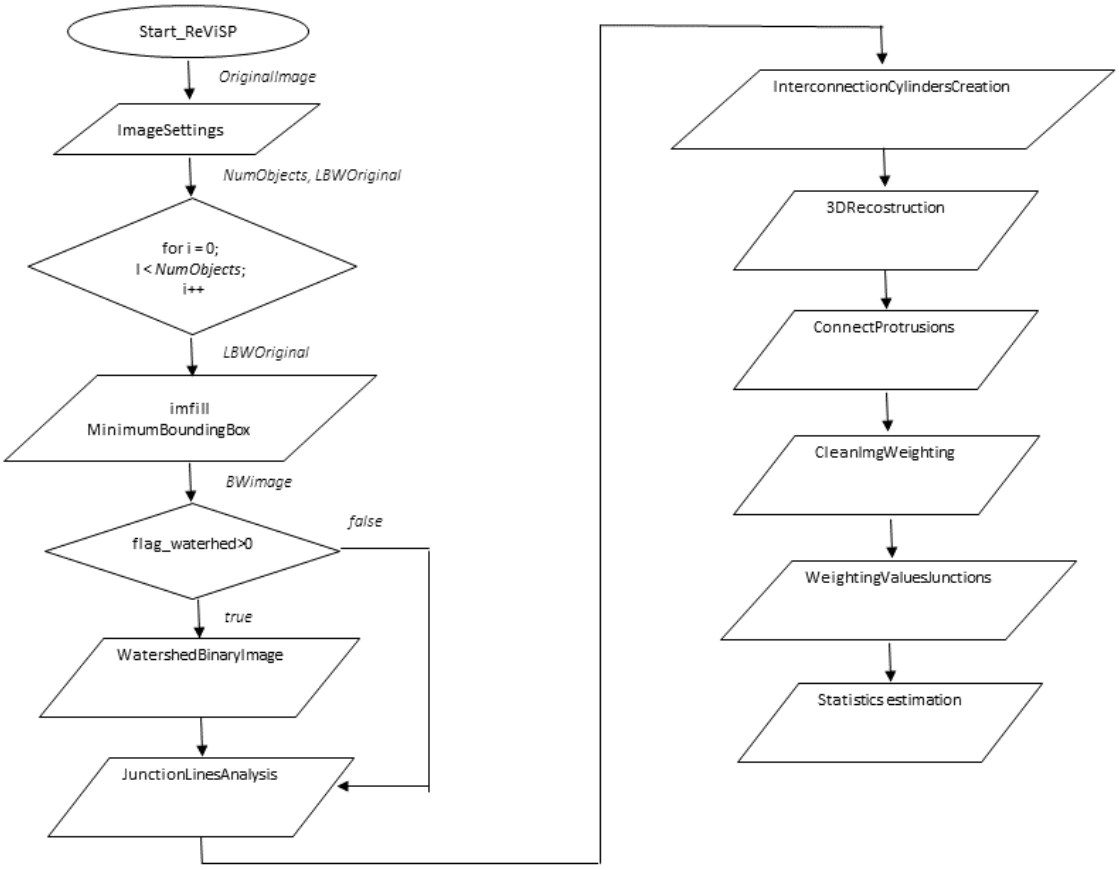
I seguenti blocchi vengono eseguiti per ogni singolo sferoide individuato nella fase precedente:

- Blocco segmentazione: si utilizza l'algoritmo di segmentazione conosciuto in letteratura con il nome di "segmentazione spartiacque", utilizzato in questo caso specifico per distinguere eventuali protuberanze nello sferoide. E' possibile, tuttavia, passare in input un parametro chiamato *'flag_watershed'* che, se uguale a 0, consente all'utente di scegliere di non effettuare questo passaggio, di fatto non andando a tenere conto di eventuali protuberanze.
- Analisi delle linee di giunzione individuate nella fase precedente: per evitare un'eccessiva segmentazione, si considerano come protuberanze separate rispetto al nucleo principale dello sferoide solo quelle collegate al nucleo da una linea di contatto inferiore a 1/3 dell'intero perimetro della sporgenza. Tale valore è stato definito

empiricamente come un buon compromesso tra rilevamento della protuberanza ed eccessiva segmentazione.

- Blocco rendering: per ricostruire il volume finale in modo accurato e realistico, non si considera semplicemente la somma delle singole parti (i.e nucleo centrale e protuberanze), ma si realizzano connessioni cilindriche aventi il diametro della base di ampiezza pari alla linea di contatto tra le parti considerate. Successivamente, sia per il nucleo centrale che per le protuberanze si calcola una mappa che riporta in ogni pixel l'altezza di quel punto rispetto al piano passante per il centro dello sferoide.
- Blocco di merging: per stimare il volume complessivo è infine necessario unire le singole parti attraverso le connessioni cilindriche precedentemente calcolate adattandone localmente l'altezza per seguire le curvature dei due oggetti da collegare. Per poter approssimare il volume nella maniera più accurata possibile si procede in 3 passaggi:
    1. Inizialmente si aggiungono alla mappa di profondità i valori delle connessioni cilindriche solo nei punti in cui non è già presente un valore;
    2. Successivamente si limita il valore massimo delle connessioni cilindriche al valore massimo degli oggetti da collegare.
    3. Infine, vengono ponderati i valori ottenuti, assegnando alla mappa di profondità il massimo tra i valori precedentemente calcolati e i nuovi valori delle connessioni cilindriche.
- Calcolo del volume come somma dei pixel non nulli della mappa di profondità ×2, supponendo quindi una perfetta simmetria sferica locale.

Di seguito viene riportato il diagramma di flusso delle funzioni di ReViSP:



3.2 Implementazione blocco scheletronizzazione

Il software ReViSP richiede in input una maschera binaria dello sferoide, o di una coltura di sferoidi. L'immagine passata in input, sebbene debba rappresentare una maschera binaria, viene interpretata da MATLAB come un'immagine RGB a 4 canali che deve essere quindi convertita in un'immagine binaria a canale singolo. Questa parte di codice è stata racchiusa all'interno della funzione *"ReViSP_ImageSettings"* che si occupa anche di eliminare dalla maschera binaria eventuali sferoidi presenti lungo i bordi, i quali infatti potrebbero inficiare i calcoli successivi. Infine tale funzione restituisce un'immagine che consente di distinguere i singoli sferoidi grazie ad una numerazione progressiva dei pixel partendo da 1 fino al numero complessivo degli sferoidi stessi.

Per quanto riguarda invece il codice C++, analogamente, l'immagine di input viene letta, sfruttando ad esempio la libreria OpenCV, come un'immagine a 3 canali GBR (notare che con la libreria OpenCV l'immagine non viene letta come RGB standard ma l'ordine dei canali

è diverso), successivamente convertita in un'immagine in scala di grigi (quindi a singolo canale) ed infine passata in input alla funzione *"ReViSP_ImageSettings"*. Questa funzione è stata ottenuta tramite il MATLAB Coder dall'omonima funzione MATLAB precedentemente descritta. Tutti i dettagli relativi alla conversione del codice ed ai problemi ad essa correlati sono descritti in dettaglio nel Capitolo 5. Di seguito è riportato il codice che esegue quanto appena spiegato e che verrà integrato nella funzione "main.cpp" del software ReViSP. La funzione *"ReViSP_ImageSettings"* generata dal MATLAB Coder richiede di passare in input l'immagine iniziale come una variabile di tipo *coder::array<unsigned char, 3U>*, per cui è quindi necessario inizializzare la variabile *BWimageOriginal* sfruttando il metodo *set()* della classe *coder::array* in modo che contenga i valori dell'immagine precedentemente letta. Per meglio capire come il MATLAB Coder gestisce le variabili array consigliamo la lettura del Libro "*Filippo Piccinini and Sofia Belloni, Guide for Dummies: from MATLAB to C++ through the MATLAB Coder, June 2021, Ed. 1. Language: bilingual edition, English and Italian. Book Publisher: YouCanPrint. ISBN: 9791220342124*" **[3]**. Non è invece necessario inizializzare le variabili di output *LBWOriginal* e *NumObjectsOriginal* in quanto ciò viene eseguito all'interno della funzione *"ReViSP_ImageSettings"*.

```
 Mat img, grayscaleImg, modifiedImg;
 coder::array<unsigned char, 3U> BWimageOriginal;
 coder:: array <double, 2U> LBWOriginal;
 double NumObjectsOriginal;

/* Read image */
 img = imread(filename, IMREAD_COLOR);
 cvtColor(img, grayscaleImg, COLOR_BGR2GRAY);

/* Init of input array */
 BWimageOriginal.set(grayscaleImg.data, grayscaleImg.cols, grayscaleImg.rows,
     grayscaleImg.channels());

/* Image conversion */
 ReViSP_ImageSettings(BWimageOriginal, LBWOriginal, &NumObjectsOriginal);
```

A partire dalla variabile *LBWOriginal* è ora possibile separare i diversi sferoidi grazie ad un semplice ciclo *for*. Di seguito si riporta un esempio di codice C++ che consente, ad ogni iterazione, di isolare nella variabile *BWimage* un singolo sferoide.

```
for (int n = 0; n < NumObjectsOriginal; n++)
{
    BWimage.set_size(rowOriginal, colOriginal);
    for (int j = 0; j < rowOriginal * colOriginal; j++)
    {
        if (LBWOriginal[j] == (n + 1))
            BWimage[j] = 1;
        else
            BWimage[j] = 0;
    }
}
```

A questo punto il singolo sferoide viene processato in modo tale da riempire eventuali buchi, intesi come pixel di valori 0 che non possono essere raggiunti riempiendo lo sfondo partendo dal bordo dell'immagine, e da ritagliare l'immagine stessa sul bordo dello sferoide.

3.3 Implementazione blocco segmentazione

La maschera binaria dello sferoide viene poi automaticamente suddivisa in sezioni comprendenti il nucleo principale dello sferoide e le eventuali protuberanze presenti. La divisione delle diverse parti è stata realizzata nel codice MATLAB originare implementando il famoso algoritmo della segmentazione con la trasformata spartiacque all'interno della funzione *"WatershedBinaryImage"*, il cui codice è poi stato modificato per consentirne la conversione in C++ tramite il MATLAB Coder, ad esempio sostituendo nell'input *varargin*, che consente di accettare un numero qualsiasi di argomento, con le singole variabili che si devono passare.

La base della *"Watershed Segmentation"* è che un'immagine a toni di grigio si possa interpretare come un rilievo topografico, in cui il livello di grigio di un pixel corrisponde all'altitudine del rilievo. Nei casi come quello in analisi, in cui l'obbiettivo è separare oggetti all'interno di una immagine binaria, per implementare questo metodo si ricorre alla *distance transform*, una trasformazione che consente di determinare per ogni pixel la distanza dal pixel diverso da 0 più vicino **[8]**. Presa quindi un'immagine di input, la si converte in immagine binaria (**Fig. 3.1a**) e successivamente si calcola la *distance transform* del complementare dell'immagine che fornisce un'immagine nera, con zone bianche in concomitanza degli

oggetti. Calcolata l'immagine in negativo di quest'ultimo output (**Fig. 3.1b**), si applica la *watershed transform*. Al termine i valori a 0 contenuti nella matrice restituita rappresentano le linee di giunzione che separano gli oggetti dell'immagine (**Fig. 3.1c**).

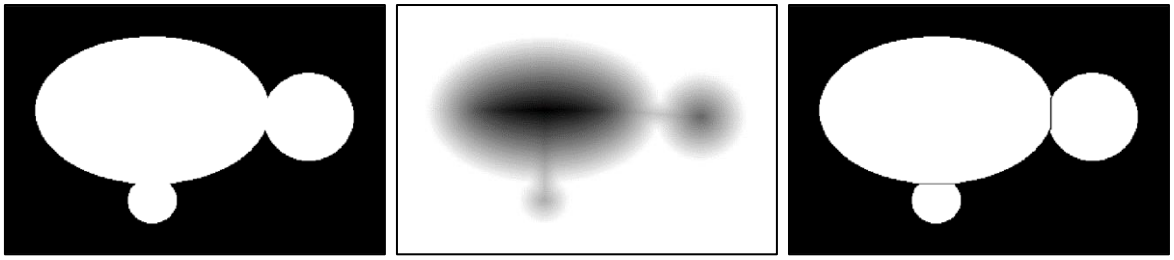


Fig. 3.1: Immagine binaria di input (**a**). Immagine ottenuta calcolando il complementare della *distance transform* del complementare dell'immagine di input (**b**) Immagine di output della w*atershed segmentation* con gli oggetti separati (**c**).

Uno dei rischi in cui si potrebbe incorrere quando si applica la *watershed segmentation* è la cosiddetta *oversegmentation*, che consiste nell'individuazione di un numero eccessivamente vasto di regioni distinte. Per tale motivo è necessario analizzare le linee di giunzione individuate nella fase precedente in modo tale da considerare come protuberanze separate rispetto al nucleo principale dello sferoide solo quelle collegate al nucleo da una linea di contatto inferiore a 1/3 dell'intero perimetro della sporgenza. Tale valore è stato definito empiricamente come un buon compromesso tra rilevamento della protuberanza ed eccessiva segmentazione. Questo controllo viene effettuato all'interno della funzione *"ReViSP_JunctionLinesAnalysis"*. Dopodiché tutti i singoli oggetti vengono ricostruiti separatamente e fusi per ottenere il rendering 3D.

3.4 Implementazione blocco rendering

Ogni sotto-parte individuata dello sferoide, rappresentata dal nucleo centrale o da una delle eventuali protuberanze, viene elaborata singolarmente con lo scopo di andare ad individuare una mappa di profondità che riporta in ogni pixel l'altezza di quel punto rispetto al piano passante per il centro dello sferoide. Questa parte di codice è stata racchiusa all'interno della funzione *"ReViSP_3DReconstruction"*, nella quale è possibile individuare alcuni passaggi fondamentali di seguito riportati.

- Rilevamento del diametro rettilineo massimo passante per il baricentro dell'oggetto;
- Rotazione dell'oggetto in modo tale da avere il diametro massimo parallelo all'asse y dell'immagine.
- La mappa 3D viene infine calcolata supponendo una simmetria sferica locale, riportando quindi in ogni punto l'altezza locale in pixel della superficie 3D rispetto al piano passante per il centro di massa dello sferoide e parallelo al piano dell'immagine. La mappa è costruita quindi adattando, per ogni posizione y, un cerchio 3D con diametro pari alla lunghezza del segmento x locale.

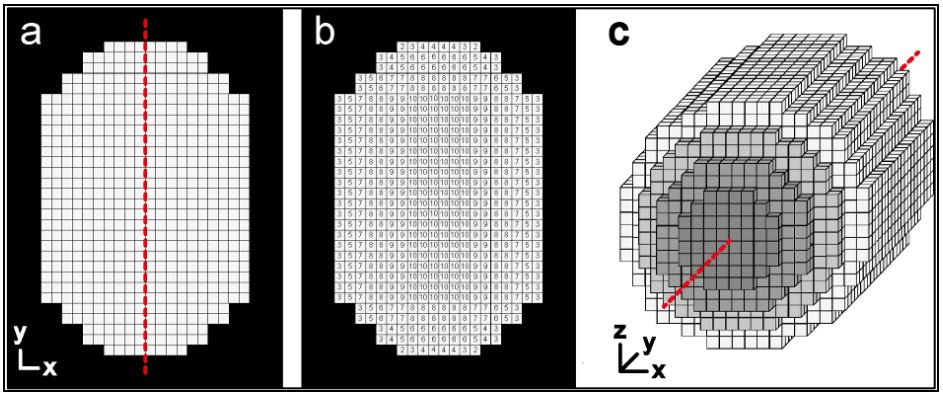


Fig. 3.2: In ReViSP la ricostruzione della superficie 3D si ottiene partendo dal posizionare l'oggetto in modo tale da avere il diametro massimo parallelo all'asse y dell'immagine (**a**). La mappa 3D x-y viene calcolata inserendo un cerchio 3D per ciascuna posizione y (**b**). La superficie 3D viene ricostruita considerando i valori della mappa 3D come altezza in pixel della metà dell'oggetto (**c**) **[1]**.

Per ricostruire il volume finale in modo accurato e realistico, non si considera semplicemente la somma del volume delle singole parti ottenuta tramite la funzione *"ReViSP_3DReconstruction"*, ma si realizzano connessioni cilindriche che hanno lo scopo di unire più fedelmente le protuberanze al nucleo centrale. Ciò viene eseguito dalla funzione *"ReViSP_InterconnectionCylindersCreation"* che per prima cosa individua i punti che rappresentano gli estremi delle linee di giunzione e successivamente 'ridisegna' queste ultime approssimandole a delle linee rette passanti per gli estremi individuati. Dopo aver effettuato le correzioni necessarie, si costruisce la mappa 2D delle connessioni cilindriche, rappresentata nella variabile *ImgWeighting*, aventi il diametro della base di ampiezza pari alla linea di contatto tra le parti considerate. E' importante notare che la versione originale di questa funzione è stata parzialmente modificata per poter essere poi processata dal MATLAB Coder

perché il codice originale sfruttava funzioni non supportate dal tool. Ad esempio inizialmente i punti di estremo delle linee congiungenti venivano individuati sfruttando la funzione MATLAB *"bwhitmiss"*, una funzione che implementa la trasformazione hit-or-miss che consente di rilevare una data configurazione (pattern) in un'immagine binaria ma che non è ancora supportata per la conversione del codice. Per questo motivo è stata quindi implementata la funzione *"BWLineLimitsFinder",* la quale consente di ottenere gli stessi risultati della *"bwhitmiss"* attraverso una successione di shift della maschera binaria stessa, e che quindi riesce ad essere convertita dal MATLAB Coder senza alcun tipo di problemi.

3.5 Implementazione blocco merging

Dopo avere eseguito il blocco di rendering descritto nel paragrafo precedente in cui sono stati ricostruiti separatamente i singoli oggetti, la cui relativa mappa 2D di profondità è rappresentata dalla variabile *DepthMapHalf* e dalle connessioni cilindriche contenute nella variabile *ImgWeighting*, è necessario fondere insieme le diverse parti per ottenere una rappresentazione 3D fedele alla realtà e quindi una stima affidabile del volume del singolo sferoide in analisi. Questa blocco di merging si compone di 3 funzioni distinte che consentono al termine di approssimare il volume in modo molto accurato:

1. Nella funzione *"ReViSP_ConnectProtrusions"* le mappe di profondità delle singole parti vengono inizialmente ricongiunte tra loro attraverso le linee di congiunzione individuate inizialmente nella maschera binaria.
2. Successivamente, la funzione *"ReViSP_CleanImgWeighting"* adatta l'altezza delle connessioni cilindriche per poter seguire le curvature dei due oggetti da collegare, limitandone quindi il valore al massimo valore calcolato in *DepthMapHalf*.
3. Infine, nella funzione *"ReViSP_WeightingValuesJunctions"* vengono ponderati i valori ottenuti, assegnando alla mappa di profondità il massimo tra i valori precedentemente calcolati e i nuovi valori delle connessioni cilindriche.

3.6 Implementazione blocco stima volumetrica

A questo punto la variabile *DepthMapHalf* riporta in ogni punto l'altezza di quel pixel rispetto al piano passante per il centro dello sferoide. Poiché i pixel di background, in cui non è quindi presente lo sferoide, hanno valore *NaN*, è possibile stimare il volume come il doppio della somma dei singoli voxel (considerando l'altezza z dei pixel corrispondente alla dimensione in x e y, considerate identiche), supponendo una perfetta simmetria sferica locale. Di seguito si riporta un esempio di codice C++ che consente di ottenere nella variabile *VolumeObj* il volume in voxel dello sferoide. In questo esempio la funzione *"rtIsNaN"* è una semplice funzione che permette di comprendere se un valore è *NaN* confrontando il valore con se stesso, in accordo con la proprietà per cui *NaN* non risulta mai uguale, maggiore o minore di altri numeri, incluso se stesso.

```
/* Statistics estimation */
 VolumePixels = 0;
 for (int j = 0; j < DepthMapHalfFinal.size(0) * DepthMapHalfFinal.size(1); j++)
     if (!rtIsNaN(DepthMapHalfFinal[j]))
         VolumePixels = VolumePixels + DepthMapHalfFinal[j];

 VolumeObj = VolumePixels * 2;
```

Al termine si restituisce il volume in voxel di ogni sferoide analizzato. Se l'immagine contiene più di uno sferoide viene mostrata anche un'immagine di output che numera progressivamente gli sferoidi in modo tale da poter associare il volume calcolato al relativo oggetto.

3.7 Assemblaggio del codice e start

Le funzioni presentate nei paragrafi precedenti sono state convertire con il MATLAB Coder e poi testate in Visual Studio singolarmente. Successivamente sono stati uniti i file sorgente e le librerie utilizzate dalle singole funzioni in un nuovo progetto Visual Studio denominato "*ReViSP*", risolvendo manualmente i conflitti che si presentavano. I dettagli relativi alle problematiche emerse in questa fase sono descritti in dettaglio nel Capitolo 5. È stata poi

realizzata la funzione “main.cpp” del software che consente di unire correttamente tutte le funzioni precedentemente testate.
È possibile eseguire il software ReViSP da linea di comando passando in input:

- il percorso dell’immagine completo del nome del file comprensivo di estensione contenente gli sferoidi di cui si vuole calcolare il volume,
- (opzionale) il percorso dell’immagine completo di nome del file comprensivo di estensione in cui si vuole salvare l’immagine di output che riporta labels indicanti l’ordine di riferimento dei volumi degli sferoidi analizzati,
- il *flag_watershed* utilizzato poi dall’algoritmo. In particolare quest’ultimo parametro consente all’utente di scegliere se tenere conto o meno delle protuberanze: nel caso in cui sia uguale a 0, infatti, non viene eseguita la *“watershed_segmentation”,* mentre con valore maggiore di 0 viene eseguito il blocco di segmentazione, ottenendo una stima del volume più accurata che tiene conto di eventuali protuberanze.

Nel caso in cui non si voglia salvare l’immagine di output, è possibile omettere il secondo parametro (per questo motivo il parametro è segnalato come “opzionale”). Si sottolinea inoltre che, nel caso in cui l’immagine contenga un solo sferoide, non essendo necessario aggiungere label per specificare a quale oggetto si riferiscano i vari volumi stimati, l’immagine di output non viene né mostrata né salvata.

Ad esempio, per analizzare l’immagine con nome “S*feroide01.png*” salvata in una cartella del Desktop dal nome “*CartellaDiProva*”, occorre aprire il prompt dei comandi di Windows e richiamare l’eseguibile *ReViSP.exe*:

1. come mostrato in **Fig. 3.3** nel caso in cui si voglia salvare l’immagine di output se nell’immagine sono presenti più sferoidi in modo da poter associare un certo volume calcolato al relativo oggetto,
2. come mostrato in **Fig. 3.4** nel caso in cui non si voglia in nessun caso memorizzare l’immagine di output.

Nell’esempio in entrambi i casi si specifica un *flag_watershed* > 0 per tenere conto di eventuali protuberanze.

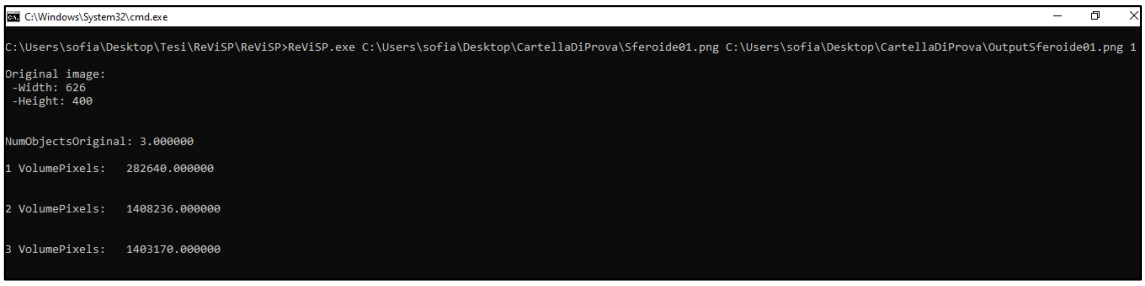


Fig. 3.3: Test "*ReViSP.exe*" dal Prompt dei Comandi nel caso in cui si voglia salvare l'immagine di output**.**

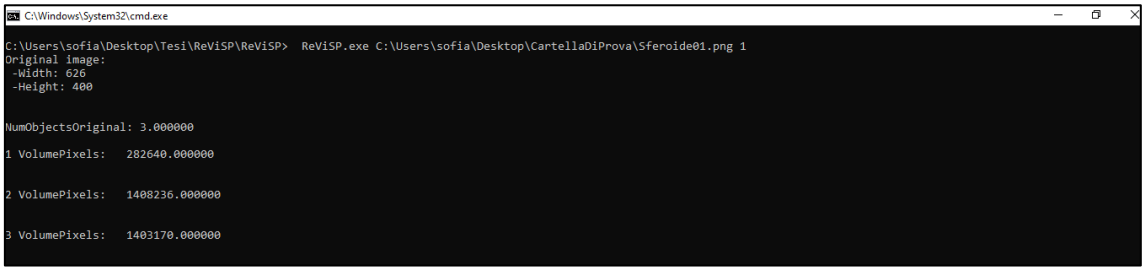


Fig. 3.4: Test "*ReViSP.exe*" dal Prompt dei Comandi nel caso in cui non si voglia salvare l'immagine di output (**b**)

È importante evidenziare come in ReViSP, sia nella versione MATLAB che in quella C++, tutte le lunghezze e le aree sono rappresentate in pixel mentre, come più volte specificato, i volumi sono in voxel. È quindi possibile, a partire dai risultati restituiti, ottenere il corrispettivo in micrometri conoscendo il coefficiente C dato dal rapporto tra una lunghezza in micrometri e la stessa lunghezza misurata in numero di pixel.

$$C = \frac{lunghezza\ in\ micrometri}{lunghezza\ misurata\ in\ pixel}$$

Dato un certo valore di conversione C, è possibile calcolare i diversi valore come segue:

- $Lunghezza\ in\ micrometri = lunghezza\ in\ pixel \times C$
- $Area\ in\ micrometri^2 = area\ in\ pixel \times C^2$
- $Volume\ in\ micrometri^3 = volume\ in\ voxel \times C^3$

CAP. 4: Validazione ed esperimenti

Una volta terminata l'implementazione del software ReViSP, il codice ottenuto è stato confrontato con l'originale MATLAB per poter comprendere quanto i risultati forniti potessero discostarsi.

Per questa fase di test sono state utilizzate 4 immagini (**Fig. 4.1**) con caratteristiche molto diverse tra loro, in modo tale da poter osservare il comportamento del software con un dataset rappresentativo.

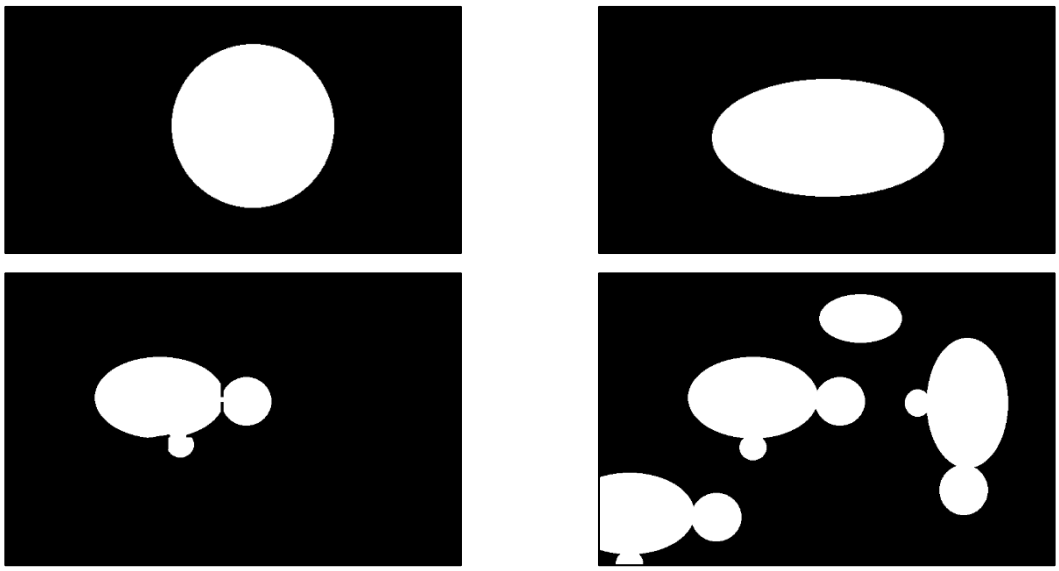


Fig. 4.1: In senso orario partendo dal lato alto sinistra: Immagine di un singolo sferoide di forma sferica (**a**). Immagine di un singolo sferoide di forma ellissoidale (**b**). Immagine di un singolo sferoide con protuberanze (**c**). Immagine di una coltura di sferoidi di cui uno tagliato lungo il bordo sinistro (**d**).

Per ciascuna di queste immagini è stato calcolato il volume degli sferoidi in essa contenuti utilizzando le tre diverse implementazioni del software ReViSP, ovvero l'implementazione MATLAB originale, l'implementazione MATLAB ottimizzata per l'utilizzo del MATLAB Coder e l'implementazione in C++. Si riportano di seguito tutti i valori dei volumi in voxel ottenuti nelle diverse misurazioni:

	Implementazione MATLAB originale	Implementazione MATLAB ottimizzata	Implementazione C++
Fig. 4.1a	5633628	5633628	5633628
Fig. 4.1b	4172152	4172152	4172152
Fig. 4.1c	2916004	2916004	2916***220***
Fig. 4.1d	Sferoide1: 586510 Sferoide2: 2979670 Sferoide3: 2982338	Sferoide1: 586510 Sferoide2: 2979670 Sferoide3: 2982338	Sferoide1: 586510 Sferoide2: 2979670 Sferoide3: 29823***72***

Tab. 1: Confronto tra le diverse implementazioni: valori in voxel ottenuti per l'analisi della singola immagine.

	Implementazione MATLAB originale	Implementazione MATLAB ottimizzata	Implementazione C++
Fig. 4.1a	223,227	213,045	210,025
Fig. 4.1b	229,723	208,108	226,936
Fig. 4.1c	506,207	536,424	468,639
Fig. 4.1d	1152,586	1096,760	866,425

Tab. 2: Confronto tra le diverse implementazioni: valori in millisecondi spesi per l'analisi della singola immagine.

Come si può notare dai dati riportati in **Tab. 1**, è emerso che le diverse implementazioni MATLAB hanno fornito sempre gli stessi risultati, mentre in alcuni casi con l'implementazione in C++ si è avuta una minima variazione, inferiore comunque dello 0,001%, riconducibile alla diversa gestione e approssimazione delle variabili nei due linguaggi (in tabella sono stati enfatizzati i valori diversi ottenuti utilizzando l'effetto grassetto-corsivo-sottolineato). I risultati ricavati possono essere quindi considerati ottimi in quanto in tutti i casi sono assimilabili ai valori ottenuti con l'implementazione MATLAB, rendendo di fatto questa implementazione del software equiparabile all'originale. Inoltre, grazie alle caratteristiche intrinseche del linguaggio C++, questa versione gode sicuramente di una maggior versatilità che la rende perfetta per poter essere integrata nei diversi framework C++ utilizzati in oncologia per la stima di feature radiomiche dei tumori. Inoltre, dai valori riportati in **Tab. 2** è possibile apprezzare come il codice risulti più performante, soprattutto all'aumentare della complessità dello sferoide in analisi.

CAP. 5: Discussione problemi incontrati

Come già anticipato, l'utilizzo del MATLAB Coder potrebbe risultare tutt'altro che immediato in quanto si presentano diverse problematiche legate sia all'ottimizzazione del codice MATLAB per consentirne la conversione, sia all'unione di tutti i file generati in un unico progetto C++. Questo tool, nella versione attuale, non consente infatti la traduzione di diverse funzione di uso piuttosto comune (ad esempio la funzione *clear*), il cui numero aumenta notevolmente se si considerano quelle per cui la conversione è possibile solo in alcuni casi particolari (ad esempio la funzione *strel* che viene correttamente convertita solo in caso di input con valori costanti). E' importante notare che nella documentazione MATLAB relativa ad ogni funzione è spesso presente un paragrafo finale relativo alla generazione di codice C/C++ in cui si specifica se la funzione è supportata o meno dal MATLAB Coder e le eventuali limitazioni.

Per quanto riguarda le difficoltà legate al codice MATLAB originale, si riportano di seguito alcuni esempi di problematiche e le modifiche apportate:

- Il codice originale conteneva un numero molto elevato di *clear* (comando che consente di eliminare una o più variabili dal workspace), molto utilizzato nella programmazione in MATLAB e che ovviamente non può essere convertito in quanto in C++ ciò non può essere fatto. In alcuni casi per evitare l'uso del comando è stato necessario l'utilizzo di una nuova variabile aggiuntiva, mentre nella maggior parte delle volte il suo uso non è stato più necessario in seguito alla divisione in funzioni del codice perché di fatto è stato rimpiazzato dallo scope delle variabili stesse.
- Prima di procedere con la generazione di codice C/C++ è importante visionare attentamente il codice per notare dei comportamenti tipici di MATLAB che non possono avere un corrispettivo in C++, ad esempio ricordandosi (*a*) che in quest'ultimo linguaggio le variabili non hanno tipo dinamico, (*b*) che è necessario pre-allocare gli array e (*c*) che le funzioni non possono avere un numero variabile di argomenti.
- Nella funzione *"ReViSP_InterconnectionCylindersCreation"* originariamente i punti di estremo delle linee congiungenti venivano individuati sfruttando la funzione MATLAB *bwhitmiss* che, come si legge nella documentazione MATLAB, si basa sulle funzioni *imerode* e *imdilate*. Queste funzioni si basano su di un oggetto *strel*

(elemento strutturante morfologico piatto, che è parte essenziale delle operazioni di dilatazione morfologica ed erosione **[9]**) passato in input, la cui conversione è supportata dal MATLAB Coder solo nel caso in cui tale oggetto sia creato a partire da un valore costante **[7]**. Poiché non era possibile ricondursi a questo caso, è stata implementata la funzione "*BWLineLimitsFinder*", la quale consente di ottenere gli stessi risultati della *bwhitmiss* attraverso una successione di shift della maschera binaria stessa, e che quindi riesce ad essere convertita dal MATLAB Coder senza alcun tipo di problemi. Sempre all'interno della funzione "*BWLineLimitsFinder*" inizialmente la base delle congiunzioni cilindriche di diametro uguale alle linee di giunzione veniva creata nel seguente modo:

```
sedisk = strel('disk', ceil(LengthLine/2), 6);
```

sfruttando la funzione *strel* per creare un oggetto strutturante di base circolare e di raggio *LengthLine/2*, dove *LengthLine* rappresenta la lunghezza della linea di giunzione in analisi il cui valore varia quindi ad ogni iterazione. Per individuare la base delle congiunzioni cilindriche è stata sfruttata la formula analitica della circonferenza per ottenere, analogamente a prima, una matrice quadrata con numero di righe e colonne corrispondente a LengthLine ed avente valore 1 in corrispondenza della base, 0 altrimenti. Di seguito si riporta il codice sostitutivo, in cui *(CentroidWB_yrow, CentroidWB_xcol)* rappresentano le coordinate del baricentro della circonferenza stessa.

```
%% (x-xc)^2 + (y-yc)^2 <= r^2
for x = 1:LengthLine
     for y = 1:LengthLine
         if  (x-CentroidWB_xcol)^2  +  (y-CentroidWB_yrow)^2  <=
r^2
             WeightingClockCircleMask(y, x) = 1;
         end
     end
end
```

Non vengono riportate nel dettaglio tutte le singole modifiche apportate in quanto sono comunque riconducibili agli esempi appena descritti, i quali possono quindi ritenersi esemplificativi delle diverse problematiche legate all'ottimizzazione del codice MATLAB per permetterne la conversione.

Per quanto riguarda invece le difficoltà legate all'unione di tutti i file generati in un unico progetto, si evidenzia quanto segue:

- C++ consente la specifica di più funzioni, chiamate *funzioni di overload*, con lo stesso nome nello stesso ambito, consentendo così di fornire una semantica diversa per una funzione, a seconda dei tipi e del numero di argomenti **[10]**. Il MATLAB Coder sfrutta molto questo principio, generando file con funzioni di supporto che sebbene presentino lo stesso nome, contengono al loro interno funzioni con un numero o tipo diverso di input a seconda delle funzioni per cui vengono generati. Per questo motivo, quando si procede ad unire tutti i file, è necessario raggruppare manualmente le funzioni omonime ma con implementazioni differenti per permettere la corretta compilazione ed esecuzione del progetto.
- Il MATLAB Coder genera codice C/C++ che potrebbe dipendere da librerie di terze parti. In questo caso specifico, siccome durante la generazione del codice l'hardware è stato impostato su "MATLAB Host Computer", vengono sfruttare le librerie TBB (Threading Building Blocks) di Intel per la programmazione parallela. Inoltre, in questo caso, poiché non diversamente specificato in fase di generazione, viene sfruttato anche OpenMP per la programmazione multi-thread che, essendo integrato nel compilatore, non richiede l'installazione di librerie esterne. Poiché le diverse funzioni in questo lavoro sono state generate singolarmente, non risultano automaticamente sincronizzate tra loro. Ciò ha creato problemi con le funzioni *"ReViSP_InterconnectionCylindersCreation"* e *"ReViSP_3DReconstruction"* che è stato possibile risolvere eseguendo entrambe le funzioni all'interno di un unico blocco, anziché in due distinti.

CAP. 6: Conclusioni e sviluppi futuri

Lo scopo principale di questo lavoro è stato quello di fornire un'implementazione in C++ ottimizzata del software ReViSP in modo tale da sfruttare la maggior versatilità del linguaggio C++ rispetto a MATLAB e consentire l'integrazione del codice in diversi framework C++ esistenti.

A tal fine è stato utilizzato il MATLAB Coder, un toolbox rilasciato dagli sviluppatori MATLAB contenente varie funzioni per semplificare la traduzione del codice tra questi due linguaggi di programmazione, il cui utilizzo tuttavia risulta spesso poco immediato anche a causa dell'assenza di una documentazione esaustiva relativa al suo utilizzo ed in particolare alle problematiche che possono emergere in fase di conversione. Tuttavia, il MATLAB Coder si è rivelato un valido strumento che ha consentito di tradurre ed ottimizzare il codice in tempi relativamente brevi: gli ottimi risultati ottenuti descritti nel Capitolo 4 sono infatti da attribuire in larga parte a questo strumento che ha permesso di "trasportare" fedelmente in un linguaggio diverso l'algoritmo originale, evitando di introdurre potenziali errori durante il processo di conversione. Inoltre, grazie alla analisi delle problematiche legate alla gestione del MATLAB Coder, il mantenimento di future versioni di ReViSP potrà essere procedura quasi immediata, consentendo di mantenere sincronizzate le implementazioni MATLAB e C++ di future release del software senza troppi problemi. Infatti un'implementazione manuale di una versione C++ di un software "giovane" come ReViSP e quindi soggetto ancora a numerose modifiche sarebbe stata molto fine a se stessa, in quanto risulterebbe utile solo fino all'uscita della nuova release. Inoltre, è importante notare che le implementazioni C++ ottenute con il MATLAB Coder sono state tutte analizzate e confrontate con quella originale al fine ottimizzare il codice ove possibile (ad esempio, dove possibile, sono state ridotte il numero di matrici interne utilizzate). Questo ha portato ad una riduzione significativa dei tempi di computazione, specialmente all'aumentare della complessità dello sferoide in analisi.

Reputiamo quindi che questo lavoro di Tesi possa essere di interesse in primis per coloro che vorranno usufruire del software di ReViSP implementato in C++, il cui codice è disponibile open-source al link indicato nel Paragrafo 2.5, ed in futuro anche per gli sviluppatori interessati a convertire in maniera semplice e veloce nuove release del codice MATLAB in codice C++.

Questa Tesi lascia spazio comunque a vari sviluppi futuri. Ad esempio nel software originale ReViSP implementato in MATLAB è inoltre presente una parte di visualizzazione della mappa di profondità dei singoli pixel, in grado di dare all'utente una visualizzazione 3D dello sferoide analizzato. Attualmente, nella versione del codice C++, non è stato implementato un modulo per la visualizzazione del rendering 3D. Si potrebbe pensare in futuro di sviluppare un'interfaccia grafica analoga a quella presente nel software MATLAB originale anche in C++, che consenta di eseguire il programma in maniera più user-friendly rispetto all'esecuzione tramite riga di comando, e che al termine dell'analisi visualizzi graficamente la struttura 3D degli sferoidi in analisi. Attualmente, infatti, il software ReViSP C++ non è dotato di interfaccia grafica a differenza della versione originale scritta in linguaggio MATLAB: sebbene si tratti di una componente non strettamente legata all'algoritmo vero e proprio, una sua implementazione potrebbe essere di aiuto agli utenti meno esperti a livello informatico ma comunque interessati all'utilizzo di ReViSP.

Bibliografia

[1] “F. Piccinini, A. Tesei, C. Arienti, A. Bevilacqua, Cancer multicellular spheroids: Volume assessment from a single 2D projection. Computer Methods and Programs in Biomedicine, 118(2):95–106, 2015. DOI: 10.1016/j.cmpb.2014.12.003”.

[2] “F. Piccinini, A. Tesei, A. Bevilacqua, Single-image based methods used for non-invasive volume estimation of cancer spheroids: a practical assessing approach based on entry-level equipment. Computer Methods and Programs in Biomedicine, 135:51-60, 2016. DOI: 10.1016/j.cmpb.2016.07.024”

[3] “F. Piccinini, S. Belloni, English and Italian Book, Guide for Dummies: from MATLAB to C++ through the MATLAB Coder, Guida per Dummies: da MATLAB a C++ attraverso il MATLAB Coder, Edition 2021”

[4] https://en.wikipedia.org/wiki/Optical_microscope

[5] https://docs.opencv.org/4.5.1/index.html

[6] https://it.mathworks.com/help/images/code-generation-for-image-processing.html

[7] https://it.mathworks.com/help/images/code-generation-with-cell-detection.html

[8] https://it.mathworks.com/company/newsletters/articles/the-watershed-transform-strategies-for-image-segmentation.html

[9] https://it.mathworks.com/help/images/ref/strel.html

[10] https://docs.microsoft.com/it-it/cpp/cpp/function-overloading?view=msvc-160#:~:text=C%2B%2B%20consente%20la%20specifica%20di,e%20del%20numero%20di%20argomenti.

Ringraziamenti

Vorrei innanzitutto ringraziare la Prof.ssa Antonella Carbonaro dell'Università di Bologna per avermi seguito come Relatrice per lo sviluppo di questa Tesi, offrendomi l'occasione di realizzare un progetto davvero interessante che unisce informatica e medicina e che mi ha permesso in questi 5 mesi di mettermi continuamente alla prova.

Ringrazio il Prof. Giovanni Martinelli dell'IRCCS IRST di Meldola, correlatore di questo lavoro, che ha permesso di svolgere al meglio questo progetto, e l'Ing. Filippo Piccinini, altro correlatore sempre dell'IRCCS IRST di Meldola che grazie ai validi consigli ed alla costanza con la quale mi ha sostenuto ma ha aiutato a risolvere le varie difficoltà.

Ringrazio anche tutta la mia famiglia che con amore, pazienza e fiducia mi ha sempre sostenuto e motivato lungo questo percorso.

Ringrazio infine tutti i miei amici di sempre, che non hanno mai fatto mancare il loro incoraggiamento e supporto, ed i nuovi amici che ho avuto la fortuna di incontrare nel corso di questi tre anni.