Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

ALGORITMI DI MESHING PER LE SUPERFICI MOLECOLARI

Elaborato in:
Computer Graphics

Relatore:
Prof.ssa
Damiana Lazzaro

Presentata da:
Roberto Lepore

Sessione Unica
Anno Accademico 2022-2023

Introduzione

Avere un'accurata rappresentazione della superficie di una molecola permette di comprenderne la natura e la funzione. In computer graphics, una superficie può essere descritta esplicitamente attraverso una *mesh*, una rete di vertici collegati in modo da formare facce poligonali, solitamente triangolari. Questa rappresentazione delle superfici è facile da visualizzare e manipolare per l'utilizzo in diverse applicazioni biotecnologiche.

La scoperta e il design di farmaci sono sicuramente tra le applicazioni più importanti in cui è fondamentale conoscere la forma delle molecole, in quanto la complementarietà tra le superfici è direttamente collegata alla forza e all'orientamento con cui due molecole possono legarsi.

In questa tesi viene analizzato il funzionamento degli algoritmi per la produzione di mesh molecolari e viene proposta *AlgoMole*, una nuova libreria C++ per lo sviluppo e l'analisi di questi algoritmi. Questa libreria permette di espandere e personalizzare facilmente algoritmi già noti oppure implementarne di nuovi, mettendo a disposizione un semplice meccanismo di configurazione di ogni fase del processo di meshing.
Il codice sorgente di *AlgoMole*, fornito di esempi e documentazione, è disponibile all'indirizzo https://github.com/rob-lepore/algomole.

Questa tesi è organizzata in 5 capitoli. Nel capitolo 1 vengono introdotte le definizioni delle diverse superfici molecolari e vengono discussi in detta-

glio alcuni dei metodi più utilizzati in ricerca. I capitoli successivi trattano del processo di sviluppo della libreria *AlgoMole*. Prima vengono esposte le tecnologie utilizzate, poi vengono analizzate le scelte di design e di implementazione di tutte le parti del progetto. Infine, vengono discussi i risultati ottenuti con gli algoritmi implementati in *AlgoMole*.

Indice

Elenco delle figure

Elenco degli algoritmi

Capitolo 1

Meshing molecolare

La forma delle biomolecole è profondamente legata alle loro funzioni e interazioni [1], per cui diventa uno strumento chiave nella risoluzione di problemi come il design di nuovi farmaci e la simulazione di interazione tra proteine.
Per calcolare la superficie di macromolecole, sono stati sviluppati sia metodi analitici che metodi numerici. I metodi analitici permettono di calcolare accuratamente area e volume della molecola, ma si rivelano poco utili in applicazioni in cui è necessario conoscere la forma locale della superficie. I metodi numerici, dei quali si tratterà in questa tesi, permettono invece di generare la superficie esplicita - i.e. la forma - delle molecole. I metodi numerici generalmente comprendono due fasi fondamentali:

1. *Space filling*: gli atomi vengono disposti su una griglia tridimensionale e si trovano quali dei suoi punti (*voxel*) appartengono al volume molecolare.

2. *Meshing*: si connettono i punti più esterni del volume molecolare con delle facce poligonali, solitamente triangoli, per creare la superficie visibile.

Per esempio, il metodo Level Set for Molecular Surfaces[2], uno dei metodi che verranno analizzati in seguito, usa la teoria degli insiemi di livello per

determinare il volume molecolare e l'algoritmo Marching Cubes per creare la superficie finale.

In questa tesi viene sviluppata **Algomole**, una libreria C++ che permette di implementare e combinare facilmente algoritmi diversi nella pipeline di meshing molecolare. Con questo strumento si potranno indagare e confrontare i metodi e gli algoritmi più comuni.

1.1 Le superfici macromolecolari

Esistono numerose definizioni di superficie macromolecolare, ciascuna delle quali risulta più utile in specifiche applicazioni. Tra le più utilizzate ricordiamo le seguenti:

- **Superficie di van der Waals** (*VDWS*): superficie che racchiude l'unione dei volumi dei singoli atomi, modellati come sfere centrate nelle posizioni degli atomi (in verde nella figura 1.1). Il raggio di queste sfere corrisponde al raggio di van der Waals, definito per ciascun elemento. La superficie così definita può descrivere correttamente la forma esterna della molecola e può essere utile per calcolare le normali alla superficie, ma non è accurata per la superficie interna, che presenta numerosi piccoli pori.

- **Superficie accessibile al solvente** (*SAS*): superficie tracciata dal centro di una sfera - che d'ora in poi chiameremo *probe* - che rotola sulla superficie di van der Waals. Il probe (cerchio nero nella figura 1.1) rappresenta una molecola di solvente, tipicamente l'acqua. Da un punto di vista geometrico, la SAS (in giallo nella figura 1.1) è equivalente alla superficie di van der Waals ottenuta aggiungendo il raggio del *probe* ai raggi degli atomi.

- **Superficie molecolare** (*MS* o *SES*): superficie tracciata dalla superficie del *probe* rivolta verso l'interno mentre rotola sulla *VDWS*. Si può notare che la superficie molecolare (parte rossa nella figura 1.1) è una superficie continua divisa in due componenti: una parte corrispondente alla *VDWS* accessibile al probe, chiamata *superficie di contatto*, e una parte chiamata *superficie rientrante* in corrispondenza dei punti in cui il *probe* è in contatto con due o più atomi.

Queste definizioni basate sul rotolamento di un *probe* sugli atomi della molecola sono state delineate da Richards in [3].

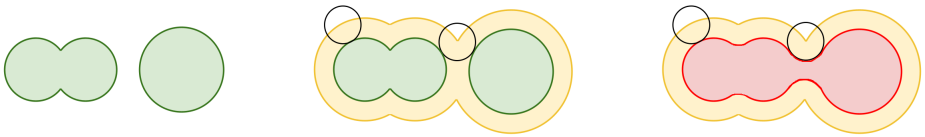

Figura 1.1: Illustrazione 2D delle tre superfici macromolecolari

1.2 Applicazioni

Conoscere e manipolare la superficie di una macromolecola come una proteina o un acido nucleico riveste fondamentale importanza in molti campi della ricerca scientifica, per esempio nel *drug discovery*[4].

Per fare in modo che un farmaco si leghi selettivamente alla molecola coinvolta in una malattia, è necessario che le due superfici siano complementari, altrimenti non sarebbe possibile portarle abbastanza vicine da creare un legame sufficientemente forte. La figura 1.2 mostra la complementarietà tra la superficie di una proteina e una piccola molecola.

Due molecole che presentano una forma simile avranno maggiore probabilità di inserirsi negli stessi siti di legame, e quindi di svolgere simili attività biologiche. Per cui trovare molecole strutturalmente simili è un elemento

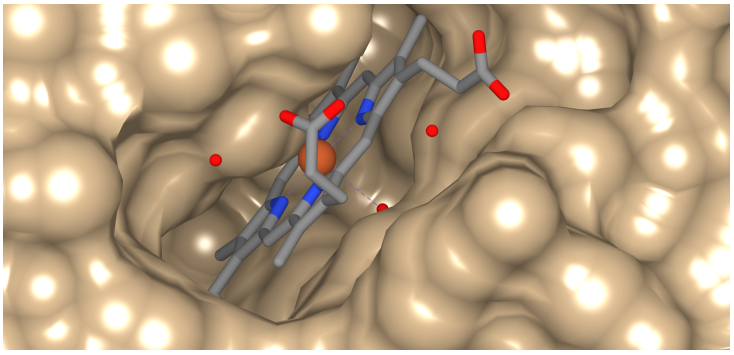

Figura 1.2: Visualizzazione nel software *UCSF Chimera* della complementarietà tra la proteina 1eca e un ligando (protoporfirina IX)

chiave nella scoperta di nuovi farmaci. Questa ricerca di molecole simili avviene attraverso una tecnica chiamata *virtual screening*: query in grandi dataset di superfici molecolari basato su metodi di computer vision e machine learning.

Il problema del *docking*, anch'esso elemento chiave nella progettazione di farmaci, consiste nel predire l'interazione tra due molecole - due proteine o una proteina e un ligando - e con quale orientamento e quale stabilità si legano per formare un complesso.
Altre applicazioni includono la progettazione di vaccini, di catalizzatori industriali, di nuovi materiali e la diagnostica medica.

1.3 Metodi analitici

Il primo algoritmo per il calcolo analitico della superficie molecolare è stato presentato da Connolly in [5]. Questo metodo calcola la superficie come composizione di sfere, archi e tori in base alle coordinate degli atomi della molecola. Successivamente è stato sviluppato l'algoritmo MSMS[6], il quale permette di calcolare sia la superficie accessibile al solvente che la superficie molecolare a partire dalla *superficie ridotta* della molecola, definita come la superficie poligonale i cui vertici corrispondo ai centri degli atomi toccati dal *probe* quando questo si trova in *posizione fissa*, cioè quando è in contatto con

tre atomi contemporaneamente. Questo algoritmo è utilizzato nel software *UCSF Chimera* per il calcolo della superficie molecolare.

I metodi analitici permettono di calcolare accuratamente i valori di area e volume della molecola, ma spesso falliscono per molecole grandi composte di numerose strutture. Inoltre, sono poco convenienti in applicazioni in cui è necessario conoscere la forma locale della superficie.

1.4 Metodi numerici

I metodi numerici per il meshing molecolare seguono generalmente un algoritmo consistente in 4 step.

1. **Pre-processing.** Gli atomi possono essere scalati e traslati in modo che l'intera molecola sia contenuta in una porzione di spazio prestabilita. Questo è necessario se si vuole contenere l'intero volume della molecola all'interno di una griglia tridimensionale di dimensione fissa.

2. **Space filling.** Viene scelto un algoritmo che trova i voxel della griglia contenuti nel solido della molecola. Se modelliamo gli atomi come sfere, possiamo trovare direttamente la VDWs e la SAS utilizzando come raggi rispettivamente il raggio di van der Waals e il raggio di van der Waals aumentato del raggio del probe scelto. Gli approcci più semplici alla soluzione di questo problema sono due: *grid-based* e *atom-based[2].* Il primo approccio attraversa ogni cella della griglia e verifica se questa si trovi o meno all'interno di un atomo. La complessità di questo approccio con N celle e m atomi è $O(N^3 \cdot m)$, eventualmente ottimizzabile a $O(N^3 \cdot \log m)$ sfruttando un *octree* nella ricerca dell'atomo più vicino. L'approccio atom-based, invece, considera un atomo alla volta e segna come interne le celle che questo occupa. Il secondo approccio, se non è necessario conoscere le celle di superficie ma solamente distinguerle tra interne e esterne, risulta molto più efficiente con una complessità $O(m \cdot k)$, dove m è il numero di atomi e k è il numero di celle occupate da un atomo in media.

3. **Surface construction.** Si utilizza un algoritmo di triangolazione per costruire una mesh corrispondente alla superficie molecolare. L'algoritmo usato più comunemente è Marching Cubes. Altri metodi che possono essere usati per creare una mesh da un insieme di punti sono Advancing Front, Marching Tetrahedra e triangolazione di Delaunay

4. **Post-processing.** La superficie può essere scalata e traslata alla dimensione e posizione originale. Inoltre possono essere applicate modifiche alla superficie per migliorarne la qualità a seconda delle necessità. Per esempio, si può applicare uno smoothing Laplaciano o un rilassamento di Voronoi, oppure rimuovere punti non necessari, suddividere la mesh aggiungendo nuovi punti o ricalcolare le normali.

Solitamente il primo step viene preceduto da una fase di parsing dei dati in ingresso, ossia file che descrivono le coordinate, i legami e le proprietà degli atomi che compongono una molecola. I formati di file più comuni sono PDB (Protein Data Bank), mmCIF (macromolecular Crystallographic Information File) e Molfile.

A seconda degli algoritmi utilizzati in ciascuna fase, si ottiene una superficie con determinate caratteristiche. La scelta di queste caratteristiche è dettata dalle necessità dell'applicazione della mesh. Potrebbe servire una superficie estremamente accurata a discapito del tempo di calcolo, oppure potrebbe essere necessario manipolare la superficie in tempo reale, nonostante questa risulti meno accurata. È quindi importante analizzare le proprietà delle superfici ottenute dai diversi algoritmi oltre che gli algoritmi stessi.

I metodi discussi in questo capitolo sono i metodi più utilizzati in ricerca e sono stati scelti perché permettono di analizzare ogni aspetto della produzione di superfici molecolari.

1.4.1 Level Set method for Molecular Surfaces

Level Set method for Molecular Surface generation (LSMS)[2] implementa un modello unico per calcolare VDWs, SAS e MS utilizzando gli insiemi di livello. Un insieme di livello di una funzione **F** è l'insieme di punti in cui la funzione assume uno stesso valore costante. Quando la funzione è in tre variabili, gli insiemi di livello si chiamano *superfici di livello* o *isosuperfici*. In questo caso la funzione rappresenta la velocità di espansione di una superficie che si espande normale a se stessa fino ad arrivare alla distanza desiderata. Quindi **F** sarà una funzione gradino che assume valore 1 prima che la superficie abbia raggiunto la distanza desiderata e 0 successivamente. La superficie trovata viene infine triangolarizzata utilizzando l'algoritmo Marching Cubes.

Fast Marching Method

La fase di *space filling* di questo algoritmo si sviluppa in due momenti:

1. propagazione della superficie verso l'esterno per ottenere la superficie di van der Waals e la superficie accessibile al solvente

2. propagazione verso l'interno per ottenere la superficie molecolare

La propagazione inizia ponendo sfere di raggio unitario in corrispondenza di ciascun atomo. Iterativamente, queste sfere vengono espanse contemporaneamente fino a quando non raggiungono il raggio desiderato (di van der Waals o accessibile al solvente).

Per implementare questo metodo, LSMS utilizza *Fast Marching Method* (algoritmo 1). Partendo dai punti della griglia interni alle sfere di origine, si espande la superficie ai punti vicini, segnando per ogni punto il momento di arrivo T.
Si utilizzano tre insiemi, inizializzati come segue: l'insieme $NEAR$ contiene le celle in corrispondenza dei centri degli atomi, l'insieme FAR contiene tutte le altre celle e l'insieme $VISITED$ è vuoto. Per ogni cella vengono segnati

l'atomo più vicino e la distanza T da esso (inizializzata all'opposto del raggio di van der Waals dell'atomo di origine). Ad ogni iterazione si seleziona la cella più vicina alla superficie in espansione e se ne guardano i vicini: quelli appartenenti all'insieme *FAR* vengono spostati in *NEAR*, mentre quelli già in *NEAR* vengono eventualmente aggiornati se raggiunti da un percorso più breve.

Questo algoritmo ha complessità $O(l \cdot \log m)$, dove l è il numero di celle all'interno della superficie accessibile al solvente e m è il numero di atomi, per cui è più efficiente dei metodi atom-based e grid-based già analizzati.

Per trovare la superficie molecolare si attua un secondo processo di propagazione simile al precedente, questa volta verso l'interno. Centrando un probe in ogni punto della superficie accessibile al solvente, i punti esclusi dalla superficie molecolare sono quelli che cadono all'interno di un probe. Quindi è possibile usare Fast Marching Method considerando come atomi i probe centrati sulla superficie accessibile al solvente. Inizialmente, l'insieme *NEAR* contiene le celle sulla SAS e l'insieme *FAR* contiene tutte le altre celle interne al volume accessibile al solvente. Questa seconda fase visita soltanto le celle interne alla superficie accessibile al solvente e solo una volta.

LSMS permette inoltre di identificare correttamente le cavità interne alla molecola, sempre utilizzando le superfici di livello.

Marching Cubes

Marching Cubes[7] è un algoritmo comunemente usato in computer graphics per la costruzione di mesh tridimensionali. Si tratta di un algoritmo semplice e molto ottimizzato, in quanto fa uso principalmente di *lookup table*. Applicazioni comuni sono la ricostruzione di una superficie da una nuvola di punti (per esempio ottenuta da scan di risonanza magnetica) o creare il contorno di una isosuperficie in un campo scalare, solitamente campionato in una griglia tridimensionale. LSMS utilizza questo algoritmo per generare la

superficie triangolarizzata che separa i punti interni al volume molecolare da quelli esterni.

In ogni cella della griglia si vogliono posizionare dei triangoli in modo da separare i vertici della cella segnati come interni da quelli segnati come esterni. Essendoci 8 vertici in una cella, le possibili combinazioni di vertici interni ed esterni sono 256. A ciascuna combinazione corrisponde una configurazione di triangoli che trovano i loro vertici sugli spigoli della cella. La posizione esatta lungo lo spigolo si trova per interpolazione lineare tra i valori dei vertici dello spigolo. Nell'implementazione di LSMS viene scelto sempre il punto medio dello spigolo. Nonostante le combinazioni siano 256, esistono solo 15 configurazioni uniche (figura 1.3).

Un problema ricorrente di Marching Cubes nasce quando una cella presenta due vertici interni o esterni diagonalmente opposti. In tal caso nasce una ambiguità, può essere corretta più di una configurazione e la scelta della configurazione sbagliata - i.e. in disaccordo con le configurazioni delle celle adiacenti - può causare la comparsa di buchi nella superficie.

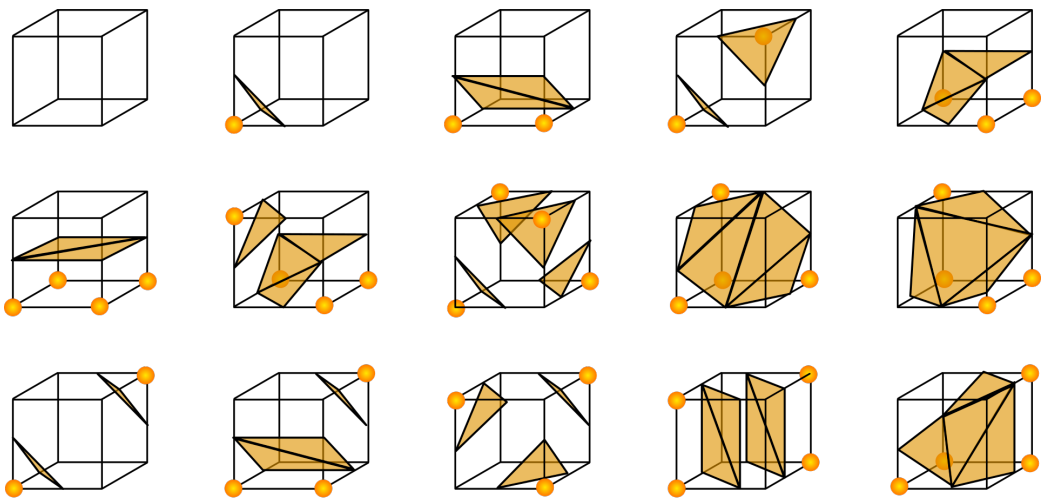

Figura 1.3: Le 15 configurazioni uniche di Marching Cubes (fonte: Wikipedia)

Algoritmo 1 Fast Marching Method

$NEAR \leftarrow$celle corrispondenti ai centri degli atomi
$T_A \leftarrow -r_A,\ \forall A \in NEAR$
$FAR \leftarrow$tutte le altre celle
$VISITED \leftarrow \emptyset$
while $NEAR \neq \emptyset$ **do**
  $VISITING \leftarrow$ cella in $NEAR$ con valore di T minore
  $NEAR \leftarrow NEAR \setminus VISITING$
  $C \leftarrow$ atomo più vicino a $VISITING$
  if $T_{VISITING} < r_{probe}$ **then**
    $NEIGHBORS \leftarrow$ vicini di VISITING
    for all $n \in NEIGHBORS$ **do**
      if $n \in FAR$ **then**
        $NEAR \leftarrow NEAR \bigcup \{n\}$
        $FAR \leftarrow FAR \setminus \{n\}$
        $T_n \leftarrow EuclideanDistance(n, C) - r_C$
      else if $n \in NEAR$ **then**
        $T_n \leftarrow \min\{T_n, T_{VISITING} + 1\}$
      end if
    end for
    $VISITED \leftarrow VISITED \bigcup \{VISITING\}$
  else
    $FAR \leftarrow FAR \bigcup \{VISITING\}$
  end if
end while

1.4.2 EDTSurf

L'algoritmo EDTSurf[8] descrive i tre tipi di superficie molecolare in termini di isosuperfici rispetto alla *trasformata distanza euclidea* (EDT). La trasformata distanza è una mappatura di ogni punto nello spazio alla sua distanza minima da un insieme di punti con una data caratteristica, per esempio dai punti esterni a un oggetto. Viene utilizzata in computer vision in algoritmi di assottigliamento[9] e di template matching[10].
Xu e Zang in [8] hanno dimostrato che la superficie molecolare può essere espressa come isosuperficie dei punti interni al solido accessibile al solvente a distanza euclidea dall'esterno pari al raggio del probe. Nonostante anche la VDWs e la SAS possano essere espresse come isosuperfici rispetto a solidi molecolari, è sufficiente calcolarle con un metodo di *space filling* più semplice (per esempio *atom-based*).

Fast 3D Euclidean Distance Transform

Una volta ottenuta la superficie accessibile al solvente con il metodo di *space filling*, si etichettano i punti della griglia in punti esterni e interni al volume. Un algoritmo utilizzato per il calcolo della trasformata distanza di un'immagine tridimensionale è *Fast 3D Euclidean Distance Transform*[11]. Questo algoritmo inizialmente trova il punto esterno più vicino ad ogni punto nel piano 2D a cui appartiene, infine utilizza questa informazione per calcolare la trasformata distanza nello spazio 3D.
Vengono definiti quattro array sulla base delle etichette dei punti (x,y) :

1. $L_z[x, y]$ contiene la coordinata x del pixel esterno più vicino a (x,y) che si trova alla sinistra di (x,y) nella stessa riga.

2. $R_z[x, y]$ contiene la coordinata x del pixel esterno più vicino a (x,y) che si trova alla destra di (x,y) nella stessa riga.

3. $B_z[x, y]$ contiene la coordinata y del pixel esterno più vicino a (x,y) che si trova sotto a (x,y) nella stessa colonna.

4. $F_z[x, y]$ contiene la coordinata y del pixel esterno più vicino a (x,y) che si trova sopra a (x,y) nella stessa colonna.

Questi quattro array possono essere condensati in due array $LR_z[x, y]$ e $BF_z[x, y]$, contenenti il quadrato della distanza tra (x,y) e il suo pixel esterno più vicino lungo l'asse x e l'asse y rispettivamente. Per ciascun piano 2D z, vengono memorizzati i quadrati della trasformata distanza di ogni punto rispetto al piano z, chiamata *trasformata distanza temporanea* ($tEDT$).

Per ottimizzare i tempi di calcolo, si sfrutta il fatto che punti adiacenti hanno una distanza dall'esterno simile, in particolare si può dimostrare che, se r_0 è la distanza minima dall'esterno del punto adiacente a (x,y), allora la $tEDT$ di (x,y) è compresa tra $(r_0 - 1)^2$ e $(r_0 + 1)^2$. Quello che viene fatto allora è visitare solo gli anelli circostanti a (x,y) dallo strato $r_0 - 1$ allo strato $r_0 + 1$. In ogni strato r_k si cerca il punto esterno più vicino a (x,y), ma misurare la distanza tra ogni coppia di punti è un procedimento dispendioso, per cui possono essere usati gli array definiti inizialmente.

Sia

$$tD = \min\{LR_Z[x, y - r_k], LR_Z[x, y + r_k], BF_Z[x - r_k, y], BF_Z[x + r_k, y]\}$$

Se $tD < r_k^2$, allora esiste un punto esterno nello strato r_k e

$$tEDT_k = tD + r_k^2$$

Per ottimizzare questo processo, possiamo evitare di cercare nello strato successivo se $tEDT_k \leq (r_k + 1)^2$ e porre $tEDT(x, y, z) = tEDT_k$.

Concluso questo processo per ogni piano 2D del volume, vengono combinate le informazioni ottenute per calcolare la trasformata distanza finale in ogni punto:

$$EDT(x, y, z) = \min\{\sqrt{(z - i)^2 + tEDT(x, y, i)}, \text{ con } i = 0, 1, \cdots, s - 1\}$$

Dato che prima vengono calcolate le trasformate distanza per ogni piano 2D dell'immagine, e solo in seguito queste vengono combinate per ottenere la trasformata 3D finale, questo algoritmo risulta facilmente parallelizzabile.

Vertex-Connected Marching Cubes

La fase di costruzione della mesh avviene utilizzando un algoritmo chiamato *Vertex-Connected Marching Cubes* (VCMC), variazione del classico Marching Cubes che, invece di prendere come vertici dei triangoli punti sugli spigoli dei cubi, prende i punti della griglia. VCMC risulta più veloce di MC e descrive una forma simile utilizzando meno vertici. Una data combinazione di vertici interni ed esterni in una cella risulta in una configurazione di triangoli vuota se contiene meno di tre vertici interni non separati da vertici esterni oppure se tutti i vertici sono interni. I vertici dei triangoli corrispondono sempre a punti interni nella griglia.

Smoothing Laplaciano

In fase di *post-processing* viene applicata un'iterazione di *smoothing Laplaciano* [12] che sposta ogni vertice nel baricentro dei vertici ai quali è collegato, in modo da ottenere una superficie più liscia e continua.
Sia N_i l'insieme dei vertici che condividono una faccia con il vertice p_i. Lo smoothing Laplaciano è un algoritmo iterativo che all'iterazione t modifica la posizione di ogni punto p_i secondo la formula $p_i^{(t+1)} = p_i^{(t)} + \lambda L(p_i^{(t)})$, dove

$$L(p_i) = \frac{1}{|N_i|} \left(\sum_{j \in N_i} p_j \right) - p_i \tag{1.1}$$

La figura 1.4 mostra le 3 superfici della molecola 1mbs calcolate con *Algomole* utilizzando l'algoritmo EDTSurf.

Algoritmo 2 Fast 3D Euclidean Distance Transform

$I \leftarrow$ punti interni
$tEDT[x, y, z] = -1,\ \forall (x, y, z)$
for $z = 0, \cdots, s-1$ **do**
  Calcolare $LR[x, y, z]$ e $BF[x, y, z],\ \forall (x, y)$
  for $x = 0, \cdots, m-1$ **do** ▷ Calcolo distanza temporanea
    for $y = 0, \cdots, n-1$ **do**
      if $(x, y, z) \notin I$ **then**
        $tEDT[x, y, z] \leftarrow 0$
      else
        $n \leftarrow$ vicino di $(x, y, z) \mid tEDT[n_x, n_y, n_z] \geq 0$
        if $\exists\, n$ **then**
          $r_0 = \sqrt{tEDT[n_x, n_y, n_z]}$
          $tEDT_k = \infty$
          for $r_k = r_0 - 1, r_0, r_0 + 1$ **do**
            $tD \leftarrow \min\{LR_Z[x, y - r_x], LR_Z[x, y + r_x], BF_Z[x - r_x, y], BF_Z[x + r_x, y]\}$
            if $tD < r_k^2$ **then**
              $tEDT_k = \min\{tEDT_k,\ tD + r_k^2\}$
              if $tEDT_k \leq (r_k + 1)^2$ **then**
                break
              end if
            end if
          end for
        else ▷ Se non si conosce la tEDT di un vicino
          $tEDT[x, y, z] = \min\{LR[x, j, z] + (y - j)^2,\ _{j=0,\cdots,n-1}\}$
        end if
      end if
    end for
  end for
end for
for all (x, y, z) **do** ▷ Calcolo della trasformata finale
  $EDT[x, y, z] \leftarrow \min\{\sqrt{(z - i)^2 + tEDT[x, y, i]} \mid i = 0, \cdots, s-1\}$
end for

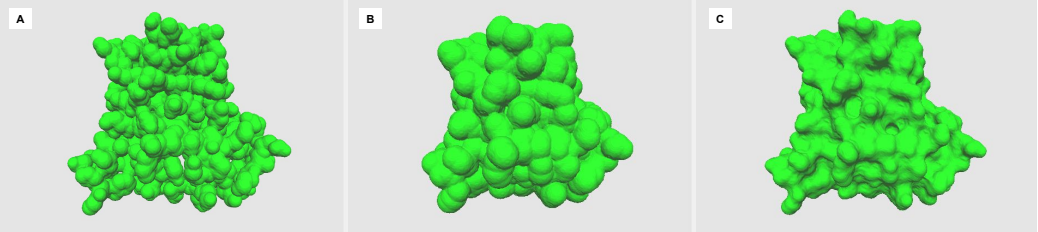

Figura 1.4: Superficie di van der Waals (A), superficie accessibile al solvente (B) e superficie molecolare (C) della molecola 1mbs calcolate con il metodo della trasformata distanza in *AlgoMole*

1.4.3 Pseudo-Gaussian Approximation

L'algoritmo Pseudo-Gaussian approximation to Lee-Richards surfaces [13] utilizza un modello di atomo diverso dai metodi discussi fino ad ora. Invece di pensare gli atomi come volumi sferici, questi vengono approssimati a una distribuzione Gaussiana tridimensionale.
Un atomo centrato in c avente N_e elettroni può essere modellato con la seguente distribuzione Gaussiana:

$$\rho(x) = \frac{N_e}{\sigma^3 2\pi\sqrt{2\pi}} \exp -\frac{||x-c||^2}{2\sigma^2} \tag{1.2}$$

Il modello a sfere, definito sulla base dei raggi di van der Waals, può essere legato a questo nuovo modello con l'utilizzo di una variabile chiamata *spread* (s), definita in modo tale che $\sigma = sr_{vdw}$. L'equazione 1.2 può quindi essere riscritta come:

$$\rho(x) = \frac{N_e}{s^3 r_{vdw}^3 2\pi\sqrt{2\pi}} \exp -\frac{||x-c||^2}{2s^2 r_{vdw}^2} \tag{1.3}$$

Definiamo ρ_0 il valore della Gaussiana in un punto a distanza r_{vdw} dall'atomo:

$$\rho_0 = \frac{N_e}{s^3 r_{vdw}^3 2\pi\sqrt{2\pi}} \exp -\frac{1}{2s^2} \tag{1.4}$$

Allora la funzione

$$\eta(x) = \rho(x)/\rho_0 = exp\left(-\frac{||x-c||^2}{2s^2 r_{vdw}^2} + \frac{1}{2s^2}\right) \tag{1.5}$$

avrà una superficie di livello di valore 1 a distanza pari al raggio di van der Waals dall'atomo. Considerando l'intera molecola invece di un solo atomo,

si ottiene una superficie che corrisponde alla superficie di van der Waals in prossimità di un solo atomo (*superficie di contatto*), mentre riempie in modo continuo gli spazi tra atomi vicini (*superficie rientrante*). Il valore dello *spread* si calcola in funzione del raggio dell'atomo e del probe che si intende utilizzare secondo la seguente formula:

$$s^2 = \frac{2r_p^2 + 2r_{vdw}r_p - r_p[4(r_{vdw} + r_p)^2 - (d)^2]^{\frac{1}{2}}}{2\ln(2)r_{vdw}^2} \tag{1.6}$$

Il valore di d, di default pari a $2r_{vdw}$, può essere fatto variare in modo da ottenere risultati diversi.

La fase di *space filling* è illustrata nell'algoritmo 3. Si considera un atomo alla volta e si somma la sua densità nei punti della griglia. Si può dimostrare che le celle più lontane di 6σ dal centro dell'atomo possono essere ignorate.

Questo procedimento ha una complessità lineare rispetto al numero di atomi e ottiene un'ottima approssimazione del volume molecolare. Infine può essere utilizzato un algoritmo di *meshing* come Marching Cubes per delineare l'isosuperficie di livello 1.

Nel caso sia necessaria una superficie di qualità più alta, è possibile estrarre gli atomi superficiali della molecola e utilizzare solo questi nel calcolo della superficie molecolare attraverso un algoritmo che segue il modello a sfere, velocizzando di molto il processo di calcolo.

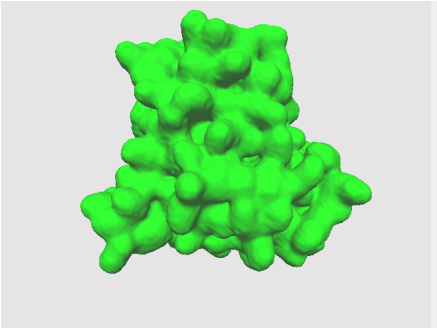

Figura 1.5: Superficie molecolare della molecola 1mbs calcolata con PGALRS in *AlgoMole*

Algoritmo 3 PGALRS space filling

for all $a \in Atoms$ **do**
  $d \leftarrow 2a_{radius}$
  $s^2 \leftarrow CalculateSpread(a_{radius}, r_{probe}, d)$
  for all $cell \in Grid$ **do**
    $\eta \leftarrow CalculateGaussian(cell, a, s^2)$
    $V_{cell} \leftarrow V_{cell} + \eta$
  end for
end for

1.4.4 Roughness-Dependent Sampling

I metodi visti finora producono superfici composte da punti uniformemente distribuiti nello spazio, ma alcune parti della superficie, in particolare aree più piane, possono essere correttamente descritte da un sottoinsieme dei punti totali. Grassmann et al. [14] hanno sviluppato un algoritmo che permette di minimizzare il numero di punti utilizzati per descrivere la molecola, preservandone la forma complessiva.
Per ogni punto i appartenente alla superficie, si considerano tutti gli n_p punti della superficie entro una distanza R_{patch} da i. Si può calcolare la curvatura della superficie in questa *patch* come la media dei coseni degli angoli tra le normali negli n_p punti e la normale media nella patch:

$$\begin{aligned} R_i =& \frac{1}{n_p} \sum_{j=1}^{n_p} \cos(\theta_{ij}) \\ \text{con } \cos(\theta_{ij}) =& \frac{\vec{v_i} \cdot \vec{v_j}}{|\vec{v_i}||\vec{v_j}|} \text{ e} \\ \vec{v_i} =& \frac{1}{n_p} \sum_{j=1}^{n_p} \vec{v_j} \end{aligned} \tag{1.7}$$

Questa misura può essere utilizzata per definire la probabilità che i punti nella patch hanno di essere presi nella superficie finale:

$$p(j) = \alpha(1 - R_i)^{\beta} \left(\frac{r_{i,j}}{R_{patch}} \right)^{\gamma(1+R_i)+\delta} \tag{1.8}$$

Questa espressione indica che i punti che hanno probabilità più alta di essere scelti sono quelli in cui la superficie è più curva e che si trovano a distanza maggiore dal centro della patch, i.e. in cui $(1 - R_i)$ si avvicina a 1 e $r_{i,j}$ è maggiore.

La scelta dei parametri α, β, γ e δ modifica il tipo di campionamento dei punti che avviene. Per esempio, se α, β e γ sono nulli si ottiene un campionamento casuale uniforme, oppure se β e γ sono nulli la probabilità non dipende dalla curvatura della superficie, ma solo dalla distanza di un punto dal centro della patch. La combinazione migliore di questi parametri, i.e. che permette di ottenere la superficie più simile a quella originale con il minor numero di punti, dipende dalla molecola in questione e può essere ottenuta risolvendo un problema di ottimizzazione che considera i *descrittori di Zernike*[15].

Capitolo 2

AlgoMole: Tecnologie

I seguenti capitoli tratteranno dello sviluppo di *AlgoMole*, una libreria statica in C++ che si pone l'obiettivo di fornire un framework completo per lo sviluppo e l'analisi di algoritmi dedicati alla produzione di superfici molecolari. Questo capitolo in particolare si concentrerà sulle tecnologie scelte per la realizzazione di questo progetto, mentre i successivi due capitoli tratteranno dell'analisi e del sviluppo della libreria.

2.1 L'utilizzo di C++

C++ è un linguaggio di programmazione noto per la sua complessità ma anche per la sua versatilità ed efficienza. Viene infatti utilizzato in numerose applicazioni in cui la gestione delle risorse e i tempi di calcolo sono critici, come lo sviluppo di sistemi operativi [16], videogiochi [17] e simulazioni di fisica [18].
Per le applicazioni che richiedono superfici molecolari con un alto grado di dettaglio, l'ottimizzazione degli algoritmi risulta fondamentale, sia relativamente ai tempi di esecuzione che allo spazio di memoria utilizzato. In primo luogo per questo motivo C++ è sembrato la scelta giusta per lo sviluppo della libreria.
Algomole non è semplicemente una raccolta di algoritmi di meshing molecola-

re già implementati, ma permette all'utente di implementare nuovi algoritmi e personalizzare ogni passo della pipeline. Per questo motivo risultano fondamentali modularità ed estensibilità, proprietà facilmente ottenibili attraverso la programmazione ad oggetti[19], paradigma supportato da C++.
Inoltre, C++ fornisce nativamente una raccolta di classi e funzioni chiamata C++ Standard Library. Questa contiene, tra le altre cose, contenitori di dati, algoritmi e componenti di gestione della memoria e gestione delle eccezioni. Esistono inoltre numerose librerie esterne ben testate e documentate, ad esempio Eigen[20] per l'algebra lineare e CGAL[21] per la geometria computazionale.
In conclusione, C++ è un linguaggio ampiamente utilizzato nello sviluppo di software scientifici e che unisce i vantaggi del controllo a basso livello delle risorse con l'astrazione della programmazione ad oggetti.

2.2 Librerie statiche

Una libreria statica è un file binario contenente classi, funzioni e variabili che viene incorporato direttamente nell'applicazione che la utilizza in fase di compilazione.
Il vantaggio principale dell'utilizzare una libreria statica è l'eliminazione dei problemi legati alle dipendenze. Questo significa che non è necessario preoccuparsi della versione delle libreria installata sul sistema finale, in quanto l'applicazione contiene già tutte le funzionalità necessarie.
Un secondo vantaggio è che le applicazioni che utilizzano una libreria statica possono essere distribuite attraverso un singolo file eseguibile autonomo, semplificando molto i processi di distribuzione e installazione. Inoltre, il linker estrae dalla libreria solamente i moduli utilizzati dall'applicazione, riducendo le dimensioni complessive dell'eseguibile finale.

2.3 Visual Studio

Visual Studio è l'ambiente di sviluppo integrato di proprietà di Microsoft. In Visual Studio si creano *soluzioni* che possono contenere uno o più *progetti*. Nel caso di *AlgoMole*, la soluzione è divisa in quattro progetti distinti:

1. **AlgoMole**: progetto contenente il codice sorgente della libreria
2. **AlgoMole-Qt**: progetto dedicato all'interfaccia grafica per visualizzare i risultati in modo interattivo
3. **AlgoMole-Test**: progetto di unit testing che consente di verificare il corretto funzionamento della libreria
4. **AlgoMole-Result**: progetto utilizzato per visualizzare ed esportare analisi sulle prestazioni degli algoritmi

Le dipendenze esterne dei quattro progetti rimangono separate, ma Visual Studio permette che la compilazione venga avviata contemporaneamente per tutti i progetti contenuti nella soluzione, facilitando uno sviluppo rapido di tutti gli aspetti della produzione della libreria.

Visual Studio mette a disposizione strumenti per il debugging che permettono di analizzare in dettaglio le prestazioni dei progetti. Questo risulta molto utile durante lo sviluppo di codice ottimizzato, per esempio gli algoritmi supportati da *AlgoMole*. Il debugger offre la possibilità di visualizzare il valore assunto dalle variabili durante l'esecuzione, fondamentale per comprendere il comportamento del codice e individuare eventuali errori, mentre il profiler mostra i tempi di esecuzione delle funzioni critiche e l'utilizzo di memoria e di CPU

Come detto precedentemente, uno dei progetti contenuti nella soluzione di sviluppo è un progetto Qt. Qt è la libreria per lo sviluppo di applicazioni con interfaccia grafica scelta per implementare la demo interattiva in cui

visualizzare i risultati degli algoritmi di meshing. Oltre ad offrire un ambiente di sviluppo dedicato, Qt permette lo sviluppo di applicazioni in Visual Studio tramite l'estensione Qt VS Tools. Questa estensione include, tra le altre cose, un ambiente di design dell'interfaccia grafica e un menù per l'installazione di moduli Qt, ad esempio per l'integrazione di OpenGL o per l'accesso alla rete.

2.4 Librerie e dipendenze

GLM (OpenGL Mathematics) è una libreria di matematica che si basa su GLSL, il linguaggio di shading di OpenGL. Viene ampiamente utilizzata nelle applicazioni di computer graphics per le trasformazioni spaziali e per il calcolo vettoriale ed è stata usata nello sviluppo di *AlgoMole* principalmente per la rappresentazione e manipolazione di vettori.

Per l'implementazione di alcuni algoritmi è stata utilizzata CGAL, una libreria di geometria computazionale che fornisce numerosi algoritmi e strutture dati per l'elaborazione geometrica. In particolare, sono risultati utili in questo progetto gli algoritmi di triangolazione e di generazione delle mesh. L'integrazione di CGAL in *AlgoMole* consente agli utenti di sfruttare funzionalità ottimizzate e collaudate, senza però vincolarli ad utilizzarle nello sviluppo dei propri algoritmi. Inoltre, vengono fornite funzioni di utilità per rendere più semplice l'interazione tra le strutture dati utilizzate da *AlgoMole* e le funzioni di CGAL.

UCSF Chimera è un potente software di visualizzazione e analisi delle strutture molecolari, incluse le superfici molecolari di cui si tratta in questa tesi. Oltre che da interfaccia grafica, Chimera è utilizzabile da linea di comando, ed è in questa modalità che *AlgoMole* richiede informazioni sulla molecola - in particolare area e volume - per confrontarle con la superficie prodotta dalla pipeline. Chimera utilizza un metodo chiamato MSMS per il calcolo

analitico - i.e. esatto - dell'area superficiale e del volume di una molecola. Questi valori vengono poi utilizzati in *AlgoMole* per misurare l'accuratezza delle mesh costruite. È necessario avere installato Chimera solamente se si vogliono utilizzare le funzioni di validazione della mesh implementate nella classe `am::utils::Validator`.

2.5 Documentazione

Per produrre una documentazione chiara, completa e facilmente consultabile, è stato utilizzato Doxygen, uno strumento di generazione automatica di documentazione dal codice sorgente. Doxygen analizza il codice sorgente ed estrae il contenuto della documentazione da commenti specifici che descrivono classi, metodi e variabili. In questo modo la documentazione è direttamente incorporata nel codice, consentendo di modificarla facilmente durante lo sviluppo e di consultarla durante l'utilizzo della libreria.
Inoltre, la documentazione può essere prodotta in una varietà di formati, tra cui HTML, PDF e LaTeX, in base alle necessità della distribuzione. La documentazione per *AlgoMole* può essere consultata al link https://roblepore.github.io/algomole-doc .

Capitolo 3

Algomole: Progettazione

Questo capitolo entra nel dettaglio degli obiettivi di *AlgoMole*, focalizzandosi sulla costruzione di un sistema robusto e in linea con i requisiti da soddisfare. Inoltre, verrà descritta l'architettura software scelta e la progettazione dettagliata delle singole componenti del sistema.

3.1 Analisi

L'obiettivo di *AlgoMole* è fornire uno strumento per lo sviluppo e l'analisi di algoritmi utilizzati nel meshing di superfici molecolari. Nel primo capitolo abbiamo individuato le quattro fasi che compongono un algoritmo di questo tipo: *pre-processing, space filling, meshing* e *post-processing.* Deve essere possibile per gli utenti scegliere o implementare gli algoritmi per ciascuna di queste fasi e inserirli nella pipeline secondo le proprie esigenze.

É quindi necessario in primo luogo riconoscere quali sono i dati in input e in output in ogni fase. Come illustrato nella figura 3.1, il flusso dei dati che abbiamo identificato è il seguente: a partire da un file che descrive la molecola desiderata, un *parser* produce una lista di atomi. Le caratteristiche degli atomi - come la loro posizione o il loro raggio - vengono poi modificate nella fase di *pre-processing.* Successivamente, la fase di *space filling* discretizza lo

spazio in cui si trova la molecola in una griglia tridimensionale di dimensione scelta e individua tutti i punti appartenenti alla griglia che si trovano all'interno del volume molecolare. L'algoritmo di *meshing* produce una superficie poligonale che divide i punti appartenenti al volume molecolare dallo spazio esterno alla molecola. Infine, nella fase di *post-processing*, vengono modificate le proprietà dei vertici della superficie in modo da migliorarne la qualità complessiva.

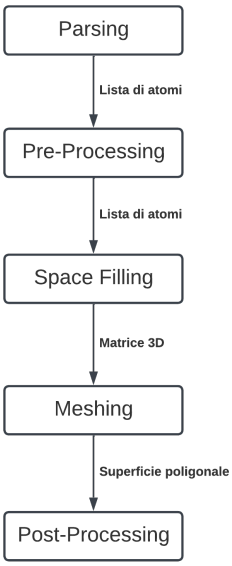

Figura 3.1: Diagramma del flusso dei dati nell'intero processo di produzione della superficie molecolare

I requisiti che nascono dal problema così definito sono i seguenti:

1. Deve essere possibile personalizzare la pipeline di meshing e configurare opzioni specifiche per ogni fase. Questo è necessario per permettere una sperimentazione flessibile degli algoritmi

2. Deve essere possibile implementare nuovi algoritmi in modo immediato. Algoritmi con lo stesso scopo possono essere usati in modo interscambiabile

3. Deve essere previsto il supporto nativo di algoritmi per ciascuna fase, a partire dal parsing dei dati molecolari da file nei formati più utilizzati

4. Deve essere possibile misurare la performance dei singoli algoritmi e le proprietà della superficie ottenuta, come area e volume. A questo scopo può essere utile supportare l'esportazione del modello 3D della molecola in formati compatibili con i software di visualizzazione e analisi

Questi requisiti delineano un'applicazione flessibile, estensibile e di semplice utilizzo nella ricerca sulle superfici molecolari.

3.2 Design

La scelta di un'architettura corretta è fondamentale per far sì che tutti gli elementi della libreria possano interagire per soddisfare i requisiti sopra elencati. In questa sezione verrà discussa l'architettura a pipeline orientata agli oggetti e verrà mostrata una possibile implementazione delle classi e strutture dati coinvolte nel processo di meshing.

3.2.1 Architettura

L'architettura scelta per *AlgoMole* è un'architettura a pipeline di oggetti[22]: è definita una sequenza lineare di componenti che prendono in ingresso un oggetto dalla componente precedente e trasmettono un oggetto alla componente successiva. Le componenti, ciascuna corrispondente a una fase del processo di meshing, dispongono di interfacce ben definite che permettono di farle comunicare tra loro e di aggiungere o sostituire componenti senza che sia necessario ridisegnare l'intera pipeline.
Una semplice architettura a pipeline di oggetti come questa - in cui la struttura della pipeline è ben definita e il flusso di dati avviene attraverso un'unica pipeline, a differenza delle implementazioni *push* o *pull* - permette modularità, flessibilità di sviluppo, efficienza di esecuzione e gestione degli errori, in particolare quest'ultima fondamentale quando si vuole permettere agli

utenti di implementare le proprie componenti. La figura 3.2 mostra l'implementazione dell'architettura in *AlgoMole*: una sola *pipe* che interagisce con le singole componenti e contiene l'intero flusso di dati.

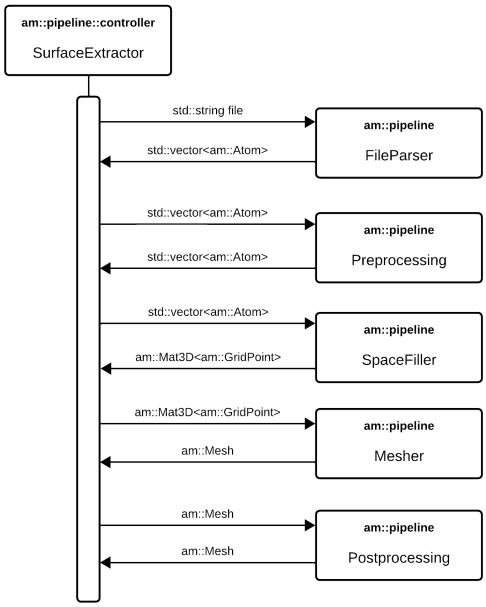

Figura 3.2: Implementazione dell'architettura a pipeline in *AlgoMole*

Dato che le componenti non comunicano direttamente tra loro, questa architettura non permette parallelismo tra le fasi. Ciononostante, le prestazioni possono essere migliorate, quando possibile, utilizzando la concorrenza all'interno delle singole componenti.

3.2.2 Classi e strutture dati

Per comprendere al meglio come sono strutturati tutti gli elementi messi a disposizione da *AlgoMole*, è necessario analizzare nel dettaglio la gerarchia dei namespace, specificando le classi e le strutture dati che contengono.

- `am`: radice di tutti gli altri namespace, non contiene nessuna costante o classe direttamente.

- `bio`: contiene informazioni riguardanti gli atomi, utili principalmente in fase di parsing.
- `gfx`: contiene funzionalità relative alla grafica e alla visualizzazione, come la struttura `Vertex` e la classe `Mesh`.
- `utils`: contiene classi di utilità, ad esempio un `Logger`.
- `math`: contiene la classe `Mat3D`, un array tridimensionale che rappresenta la discretizzazione dello spazio necessaria nelle fasi di *space filling* e *meshing*.
- `exceptions`: contiene le classi di errore specifiche per la pipeline di meshing.
- `pipeline`: contiene le funzionalità più importanti della libreria, prime tra tutte le implementazioni delle componenti per ogni fase della pipeline.
  - `options`: contiene le opzioni non numeriche con le quali poter personalizzare i calcoli svolti dalle componenti, come la modalità di colorazione della mesh e il tipo di calcolo delle normali da eseguire.
  - `controller`: contiene la pipeline che controlla il flusso dei dati (classe `SurfaceExtractor`) e un *builder* che permette di configurare la pipeline in modo semplice.

La costruzione della pipeline richiede all'utente di impostare tutte le componenti e configurare le opzioni necessarie. Per semplificare questo processo, è stato implementato un *builder pattern* (figura 3.3) che permette di configurare in modo chiaro la pipeline e ottenere direttamente la pipeline corrispondente agli algoritmi supportati. Il builder pattern è un design pattern creazionale che risolve il problema della creazione di rappresentazioni differenti di oggetti complessi, nel nostro caso la pipeline. Può essere utilizzato un `Director` per creare direttamente un builder per un metodo desiderato, altrimenti si può creare un `SurfaceExtractorBuilder` e configurare ogni fase ed opzione

manualmente. Un esempio di utilizzo del `Director` per ottenere la pipeline del metodo PGALRS, di cui abbiamo parlato nel Capitolo 1, è il seguente:

```
1.  using namespace am::pipeline::controller;
2.  SurfaceExtractorBuilder builder;
3.  Director director;
4.  director.make(Director::Type::GAUSSIAN);
5.  SurfaceExtractor se = builder.build();
6.  am::gfx::Mesh* mesh = se.generateSurfaceMesh(pdb);
```

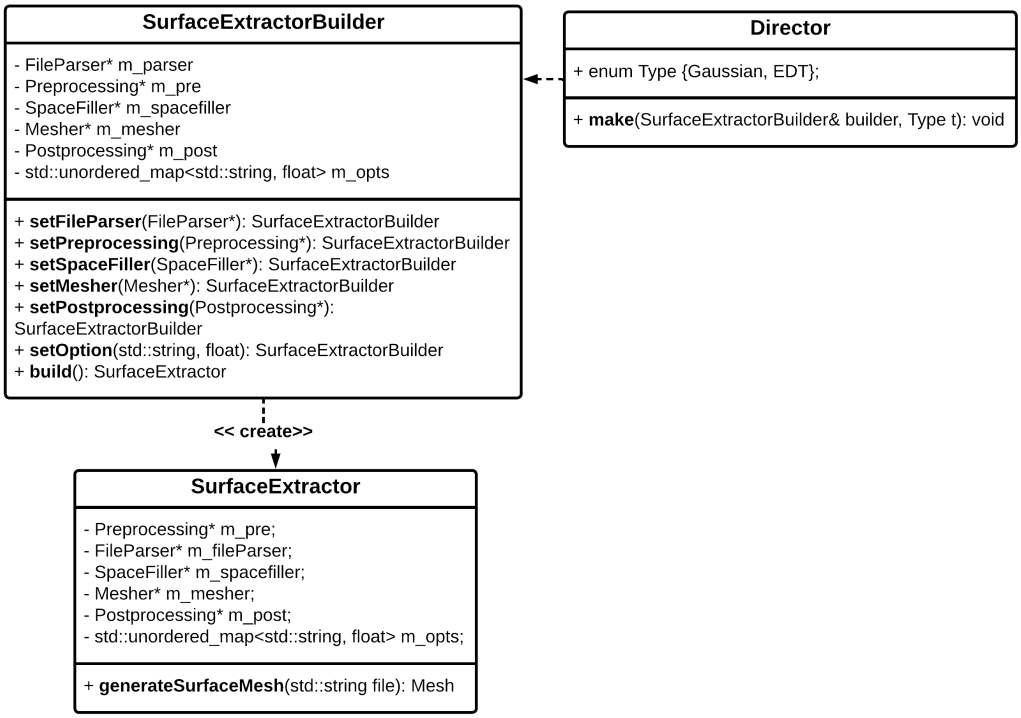

Figura 3.3: Diagramma delle classi UML raffigurante il builder pattern

Il `SurfaceExtractor`, che come già detto rappresenta la pipeline, contiene riferimenti alle implementazioni da eseguire per ciascuna fase (figura 3.4). Le componenti devono estendere l'interfaccia della fase a cui corrispondono, scegliendo tra `FileParser`, `Preprocessing`, `SpaceFiller`, `Mesher` e

`Postprocessing`. Oltre ai dati da processare, le componenti prendono in ingresso anche una *mappa* di opzioni. Le opzioni possono essere specificate in fase di configurazione attraverso il metodo `setOption(std::string key, float value)` del builder.

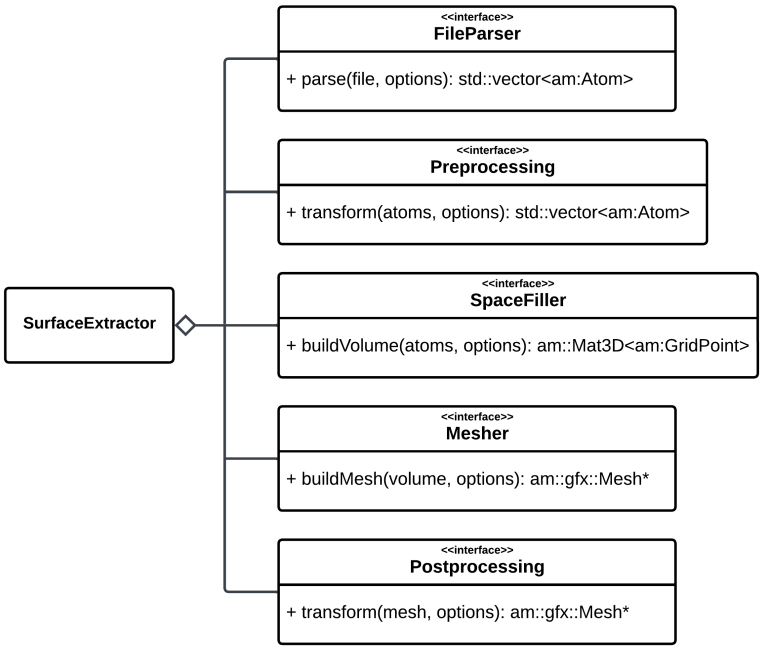

Figura 3.4: Diagramma delle classi UML della composizione della pipeline

Come mostrato in figura 3.2, i dati che vengono scambiati tra le componenti vengono rappresentati con classi e strutture dati definiti in *AlgoMole* (figura 3.5). La struttura `Atom` contiene posizione, raggio, catena e simbolo dell'elemento di un atomo come descritto nel file di input. Oltre che all'interno di vettori nelle prime fasi della pipeline, la struttura `Atom` è utilizzata anche all'interno della struttura `GridPoint`, che infatti contiene un `Atom`, relativo all'atomo più vicino, e un valore numerico. La classe `Mesh` è un'implementazione *indexed face* di una mesh: contiene un vettore di vertici (struttura `Vertex`) e un vettore di indici che fanno riferimento a vertici nel primo vettore. Tre indici consecutivi indicano i vertici che descrivono una faccia della superficie. Oltre a questi due vettori, la classe `Mesh` contiene una

tabella di adiacenza per ottenere efficientemente i vertici collegati a un dato vertice. Inoltre fornisce metodi per l'esportazione in formato OBJ e per il ricalcolo delle normali. Un vertice contiene le informazioni utili al rendering della superficie, in particolare posizione, colore e normale.

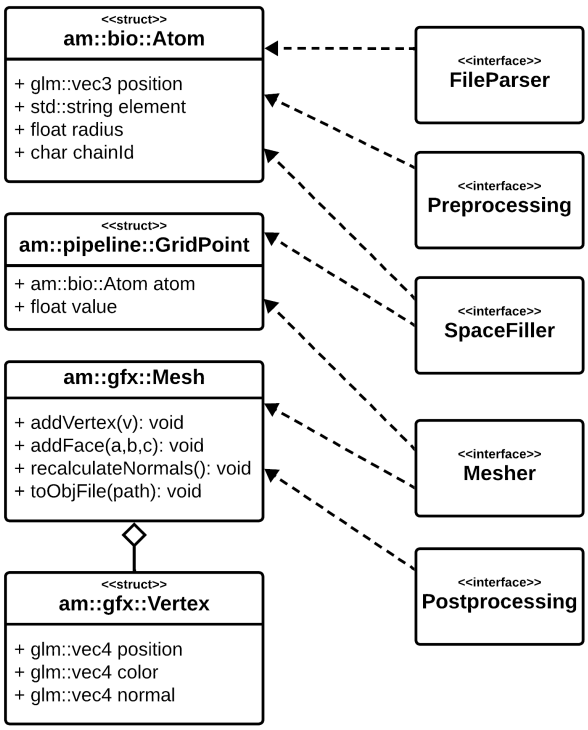

Figura 3.5: Diagramma delle dipendenze tra componenti e strutture base

Capitolo 4

Algomole: Sviluppo

Questo capitolo si focalizza sull'implementazione degli algoritmi e degli strumenti di validazione dei risultati inclusi nella libreria *AlgoMole*.

4.1 Implementazione degli algoritmi

4.1.1 Parsing

Lo scopo di un algoritmo di *file parsing* è quello di estrarre informazioni da un documento di testo e strutturarle nel modo desiderato. Per esempio, un parser potrebbe essere impiegato per valutare il risultato di una espressione matematica o per verificare la correttezza sintattica di un programma.

In *AlgoMole*, gli algoritmi di parsing devono estendere l'interfaccia `FileParser` e implementare il metodo `parse(file, options)`.
Algomole supporta nativamente il parsing di file in formato Protein Data Bank (PDB) attraverso la classe `PdbFileParser`. I file PDB contengono le coordinate degli atomi che compongono una struttura molecolare. In particolare, ogni riga contenente un dato di tipo `ATOM` riporta l'elemento dell'atomo, l'amminoacido a cui appartiene e le sue coordinate nello spazio in *Angstrom* ($1 \cdot 10^{-10}$ metri).

Il processo di parsing è molto semplice. Da ogni riga che inizia con la parola ATOM si leggono le informazioni utili al meshing, cioè le coordinate, l'elemento e la catena dell'atomo. Inoltre, si ricava il raggio di van der Waals corrispondente all'elemento dell'atomo. Come valori dei raggi di van der Waals sono stati presi quelli utilizzati da *UCSF Chimera*. Con questi dati si costruisce una variabile di tipo `Atom` e la si inserisce in un array che viene restituito una volta terminato il file in ingresso.

4.1.2 Space Filling

Gli algoritmi di *space filling* discretizzano una regione di spazio in un insieme di punti ai quali viene assocciato un valore numerico.
Per rappresentare questa discretizzazione in *AlgoMole* si utilizza la classe `Mat3D<GridPoint>`, la quale mantiene i dati in un semplice `std::vector` unidimensionale. Il metodo `Mat3D::at(x,y,z)` restituisce l'elemento contenuto nella posizione specificata, la quale viene trasformata nell'indice corrispondente nell'array unidimensionale sottostante secondo la formula:

$$indice = x + y * width + z * width * height$$

Approccio Grid-based

Uno dei metodi più semplici per etichettare i punti dello spazio (discretizzato) interni al volume molecolare consiste nello scorrerli e verificare se ricadano all'interno del raggio di almeno un atomo.
Questo approccio ha una complessità temporale $O(L^3 \cdot n)$, dove L è la lunghezza dello spigolo della griglia e n è il numero di atomi della molecola. La ricerca dell'atomo più vicino può essere ottimizzata utilizzando strutture dati come *octree* o *R-tree* per raggiungere una complessità totale $O(L^3 \cdot \log n)$.
Questo algoritmo è stato implementato all'interno di *AlgoMole* nella classe `am::pipeline::GridSpaceFiller`.

Approccio Atom-based

Un'ottimizzazione immediata all'approccio precedente considera un atomo alla volta ed etichetta i punti dello spazio circostanti come interni al volume. Dato che la maggior parte degli atomi ha lo stesso raggio, è possibile memorizzare la disposizione dei punti interni a un atomo e replicarla intorno a tutti gli atomi con uguale raggio. Questo algoritmo ha una complessità temporale $O(n \cdot l)$, dove n è il numero di atomi e l è il numero medio di punti interni al volume di un atomo.
Questo algoritmo è stato implementato all'interno di *AlgoMole* nella classe `am::pipeline::AtomSpaceFiller`.

Sia l'approccio grid-based che l'approccio atom-based non sono adatti al calcolo della superficie molecolare, ma solo della superficie di van der Waals e della superficie accessibile al solvente.

Euclidean Distance Transform

Come descritto nel capitolo 1, l'algoritmo di *space filling* utilizzato da *EDTSurf* calcola la superficie molecolare come trasformata distanza del volume accessibile al solvente, ottenibile da uno dei metodi visti precedentemente.
Il metodo della trasformata distanza (algoritmo 2) necessita di diverse strutture di supporto durante l'esecuzione. In particolare, data una matrice $n \times m \times s$ che rappresenta il volume accessibile il solvente, vengono utilizzate:

- una matrice $n \times m \times s$ di `float` contenente la trasformata distanza temporanea di ciascuno strato della matrice originale

- una matrice $n \times m \times s$ di `float` contenente la trasformata distanza della matrice originale

- 6 matrici $n \times m$ di `float` per ciascuno strato della matrice originale

Per cui la complessità spaziale di questo algoritmo è $O(n \cdot m \cdot s)$, quindi lo spazio richiesto durante l'esecuzione non dipende dalla grandezza della molecola, ma solo dalla risoluzione della discretizzazione dello spazio.

L'implementazione effettuata nella classe `am::pipeline::EDTSpaceFiller` permette l'esecuzione parallela della fase di calcolo della trasformata distanza temporanea. È possibile selezionare il numero di thread da utilizzare attraverso l'opzione `filling_threads`.

Pseudo-Gaussian Approximation

Per calcolare l'approssimazione pseudo-Gaussiana della superficie molecolare, sono richiesti ripetuti calcoli computazionalmente dispendiosi. In particolare, per ogni atomo è necessario calcolare s^2 e σ come definiti nella sezione 1.4.3. Si può notare però che questi valori dipendono unicamente dall'elemento dell'atomo in questione, per cui sono state utilizzate due mappe che vengono riempite durante l'esecuzione contenenti i valori di s^2 e di *sigma*.

4.1.3 Meshing

Il prodotto finale della pipeline è una mesh corrispondente alla superficie della molecola. L'implementazione della classe `am::gfx::Mesh` è stata fatta seguendo una rappresentazione *indexed face*, che descrive la mesh utilizzando due array, uno per i vertici e il secondo per le facce poligonali. Per ottimizzare alcune operazioni necessarie sulla mesh, è stata aggiunta una mappa di adiacenza dei vertici, della quale benificiano soprattutto gli algoritmi di *post-processing*.

Marching Cubes

Marching Cubes è uno degli algoritmi di ricostruzione della superficie più utilizzati in computer graphics.
Le prestazioni di questo algoritmo sono ottimizzate attraverso l'utilizzo di

una *lookup-table* con cui è possibile ricavare la posizione dei vertici in una cella della griglia in tempo costante. La posizione della triangolarizzazione nella *lookup-table* è univocamente identificata dalla configurazione di vertici interni ed esterni nella cella. Il codice che calcola questa corrispondenza è il seguente:

```
1.  int configIndex = 0;
2.  if (grid.at(x,y,z).value < isolevel)          configIndex |= 1;
3.  if (grid.at(x+1, y, z).value < isolevel)      configIndex |= 2;
4.  if (grid.at(x+1, y, z+1).value < isolevel)    configIndex |= 4;
5.  if (grid.at(x, y, z+1).value < isolevel)      configIndex |= 8;
6.  if (grid.at(x, y+1, z).value < isolevel)      configIndex |= 16;
7.  if (grid.at(x+1, y+1, z).value < isolevel)    configIndex |= 32;
8.  if (grid.at(x+1, y+1, z+1).value < isolevel) configIndex |= 64;
9.  if (grid.at(x, y+1, z+1).value < isolevel)    configIndex |= 128;
```

All'interno della *lookup-table* sono memorizzati array di 16 interi che indicano in quali spigoli della cella inserire i vertici della triangolarizzazione.

L'algoritmo accede alla *lookup-table* una volta per cella e ogni cella viene visitata una sola volta, quindi ha complessità temporale $O(L^3)$.

Alpha Wrapping

In computer graphics, si definisce Alpha Shape di un insieme di punti come una famiglia di politopi che possono essere convessi e disconessi. Quando il parametro α vale ∞, la forma corrisponde all'inviluppo convesso dei punti, mentre al decrescere di α compaiono rientranze sulla superficie che possono anche arrivare a unirsi per creare tunnel e buchi.

L'implementazione di questo algoritmo in *AlgoMole*, presente nella classe `am::pipeline:AlphaWrapMesher`, utilizza la funzione `alpha_wrap_3` della libreria CGAL per ottenere l'Alpha Shape dei punti interni al volume

molecolare. Le opzioni relative a questo metodo sono `relative_alpha` e `relative_offset`, che hanno valori di default 100 e 500 (corrispondenti a $\alpha = 1/100$ e $offset = 1/500$). Il valore di offset controlla la distanza dei vertici della mesh dai punti originali: un valore basso produce una mesh più simile alla forma obiettivo ma anche più densa.

4.1.4 Post-processing

Il processo di *post-processing* può migliorare notevolmente la qualità della mesh ottenuta, sia per quanto riguarda l'occupazione di memoria che per la precisione dell'area e del volume. Questa fase non modifica sostanzialmente la forma della molecola, per cui in certe applicazioni può essere tralasciata (componente `NonePostprocessing` in *AlgoMole*).

Laplacian Smoothing

Il processo di smoothing Laplaciano sposta ogni vertice della mesh nel baricentro dei vertici ai quali è collegato. La disponibilità di una mappa di adiacenza per i vertici all'interno della mesh permette di non dover scorrere tutte le facce per ogni vertice, quindi la complessità temporale si riduce a $O(n \cdot c)$, dove n è il numero di vertici e c è il numero medio di vertici connessi per ciascun vertice.

Capitolo 5

Algomole: Risultati

In questo capitolo vengono discussi i metodi di validazione dei risultati ottenuti e viene mostrata una demo con interfaccia grafica come esempio di applicazione di *AlgoMole*.

5.1 Validazione

Una prima analisi sui risultati della pipeline può essere effettuata con l'utilizzo della classe `am::utils::Validator`, la quale permette di calcolare gli errori relativi di area superficiale e volume della mesh. Come valori esatti per il confronto vengono prese le misurazioni di *UCSF Chimera*.

I file utilizzati in input sono stati presi dal *RCSB Protein Data Bank*, uno dei più importanti archivi di strutture tridimensionali di grandi molecole memorizzate in formato PDB.

Nella valutazione dei risultati è stato usato un dataset di 27 molecole di lunghezza compresa tra i 454 e i 3472 atomi. Misurare l'errore relativo medio su area e volume della pipeline su un dataset come questo permette di valutare se la precisione è sufficiente per l'applicazione finale. Nella figura 5.1 sono mostrati gli errori relativi di area e volume per diverse combinazioni di algoritmi nella pipeline con dimensione della griglia 254^3 e raggio del probe

1.4 Å. È necessario notare che i risultati dell'algoritmo di alpha wrapping dipendono dalla scelta dei parametri α e *offset*, i quali valori ottimi possono variare a seconda della specifica molecola.

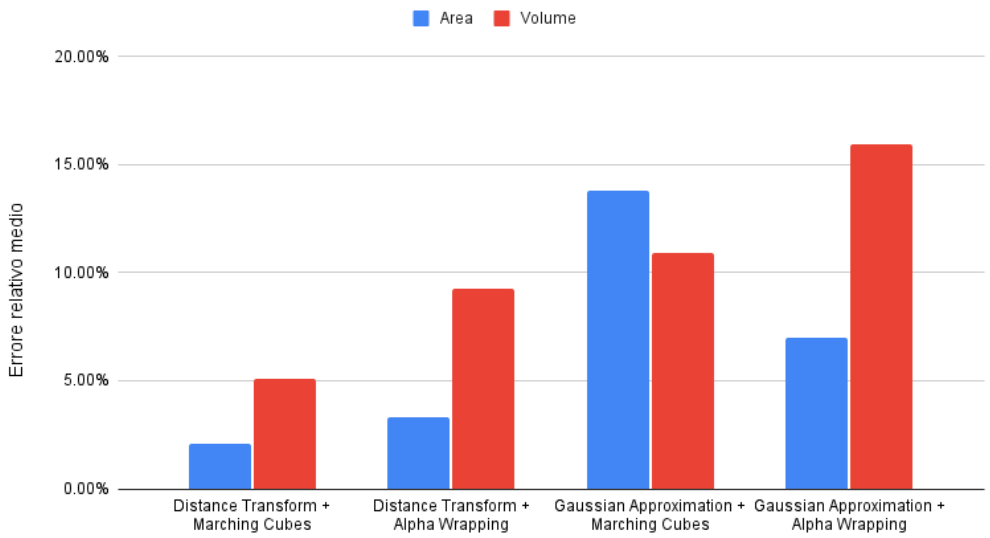

Figura 5.1: Errori relativi medi del calcolo di area e volume

Un'ulteriore analisi che può essere fatta sul valore di area e volume della mesh consiste nel verificare che la qualità della mesh aumenti all'aumentare della risoluzione (figura 5.2). Questa verifica è utile soprattutto in fase di implementazione degli algoritmi. Nella figura 5.3 si può notare la tendenza degli algoritmi ad avvicinarsi all'area analitica con l'aumentare della dimensione della griglia.

Infine, sono state misurate le prestazioni degli algoritmi con l'utilizzo degli strumenti di profilazione di Visual Studio (figura 5.4). In questo caso è utile ordinare il dataset per numero di atomi che compongono le molecole. Nelle precedenti analisi non sono state considerate molecole di dimensioni eccessive perché *UCSF Chimera* non riesce a calcolarne area e volume. Nell'analisi delle prestazioni, invece, includere molecole di grandi dimensioni permette di verificare la correttezza della complessità prevista.

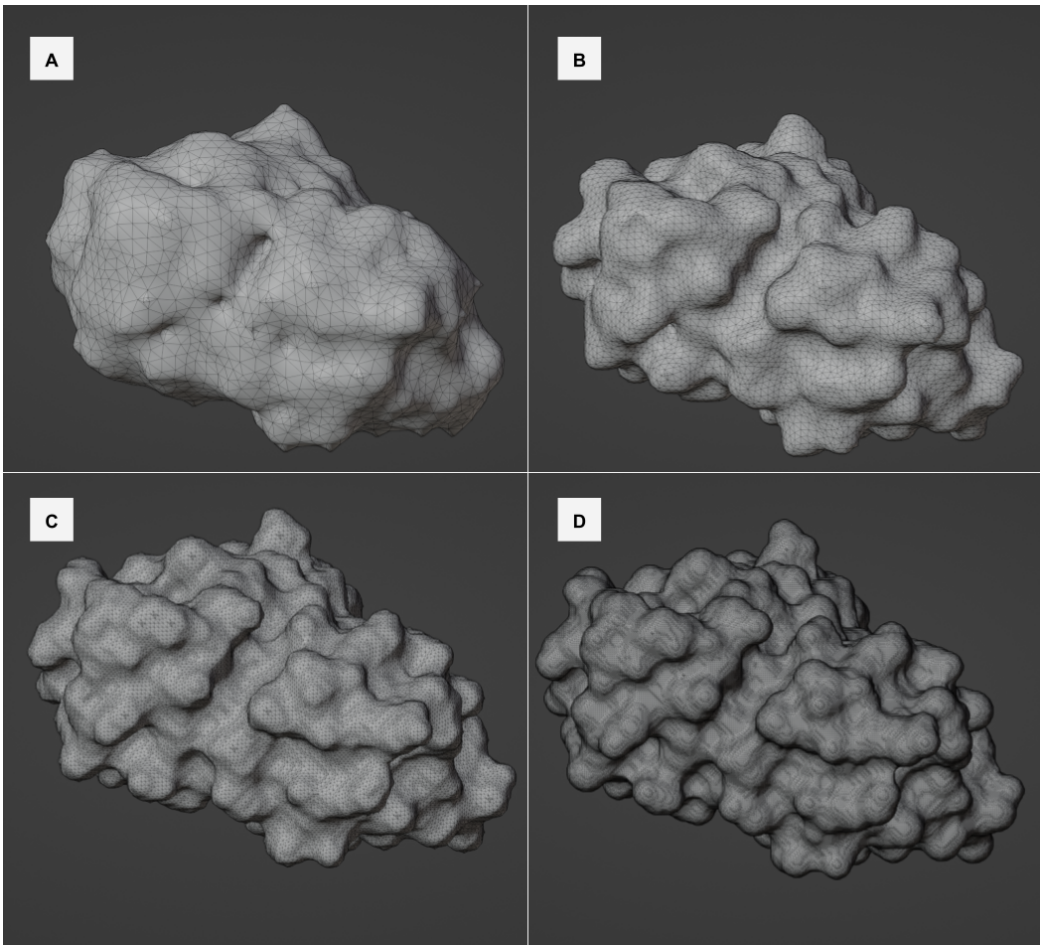

Figura 5.2: Aumento della definizione della mesh all'aumentare della dimensione della griglia: 32^3 (A), 64^3 (B), 128^3 (C) e 256^3 (D)

.

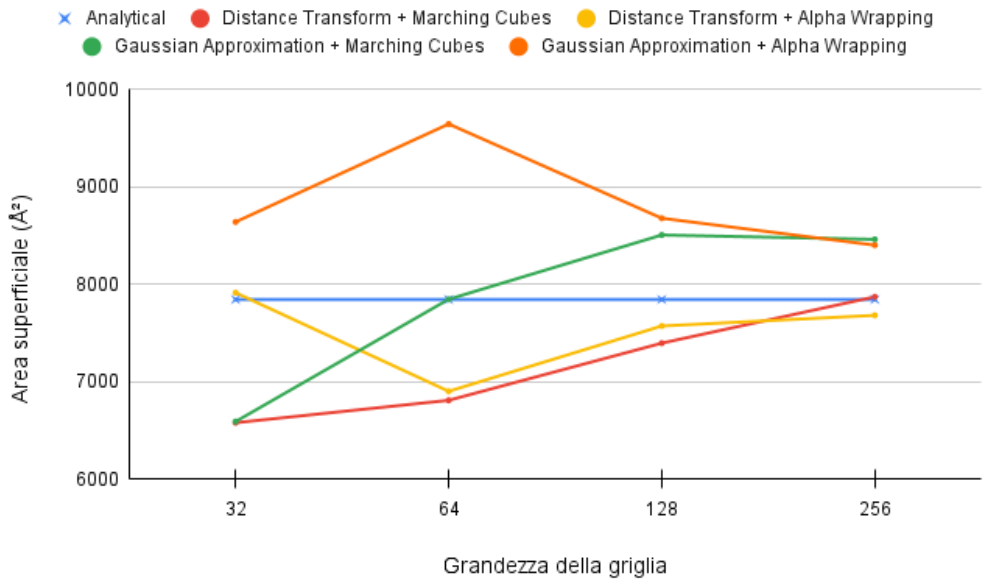

Figura 5.3: Confronto delle aree superficiali ottenute dagli algoritmi a diverse risoluzioni per la molecola 1mbs

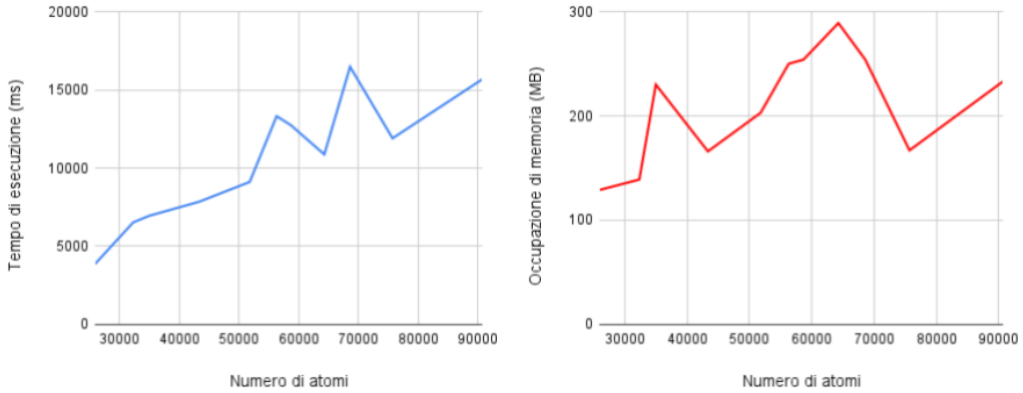

Figura 5.4: Grafici del tempo di calcolo (a sinistra) e dell'occupazione massima di memoria (a destra) dell'algoritmo della trasformata distanza su un set di molecole grandi

5.2 Esempio applicativo

Come esempio applicativo, è stata implementata una demo grafica che offre una dimostrazione visiva delle differenze tra i risultati ottenuti utilizzando algoritmi diversi.

In questa interfaccia utente (figura 5.5) è possibile selezionare gli algoritmi di *space filling* e di *meshing* da utilizzare e impostare i valori per il raggio del *probe* e la dimensione della griglia. Gli algoritmi a disposizione sono quelli discussi nel capitolo 4: `Euclidean Distance Transform` e `Gaussian Approximation` per lo *space filling*, `Marching Cubes` e `Alpha Wrapping` per il *meshing*. La figura 5.6 mostra i risultati ottenuti dalle possibili combinazioni di questi algoritmi
Nel box di input è possibile inserire l'ID della molecola che si vuole visualizzare. Il file in formato PDB corrispondente viene automaticamente scaricato dal database *RCSB PDB* e utilizzato come input della pipeline di *AlgoMole*.

Una volta calcolata la superficie molecolare, questa viene mostrata nella parte destra dell'interfaccia. La scena 3D è renderizzata con OpenGL, direttamente compatibile con le strutture di *AlgoMole*. Con l'utilizzo del mouse e della tastiera, è possibile ruotare e spostare la molecola (figura 5.7). Nell'area di testo in basso a sinistra vengono visualizzati i valori di area superficiale e volume della mesh con i corrispondenti errori relativi.
Il sistema di colorazione della molecola è stato configurato per evidenziare distintamente le diverse catene che compongono la molecola complessiva (figura 5.7), tuttavia *AlgoMole* supporta anche la colorazione per atomo e l'utilizzo di un unico colore per l'intera superficie.

Questa demo grafica è quindi uno strumento utile per valutare le differenze visive tra i vari algoritmi in modo intuitivo e veloce.

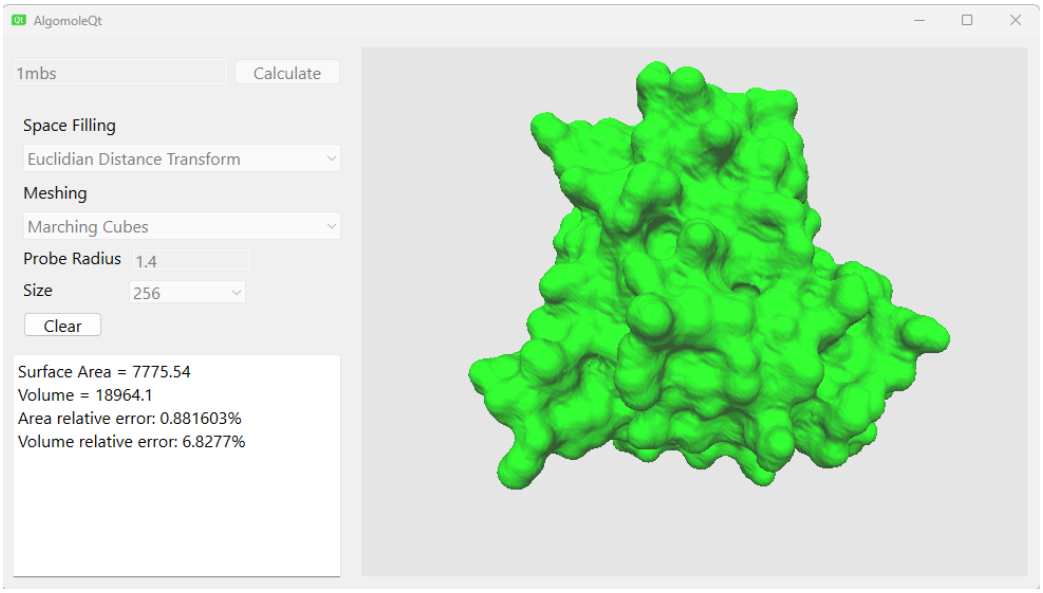

Figura 5.5: Interfaccia grafica della demo di utilizzo di *AlgoMole*

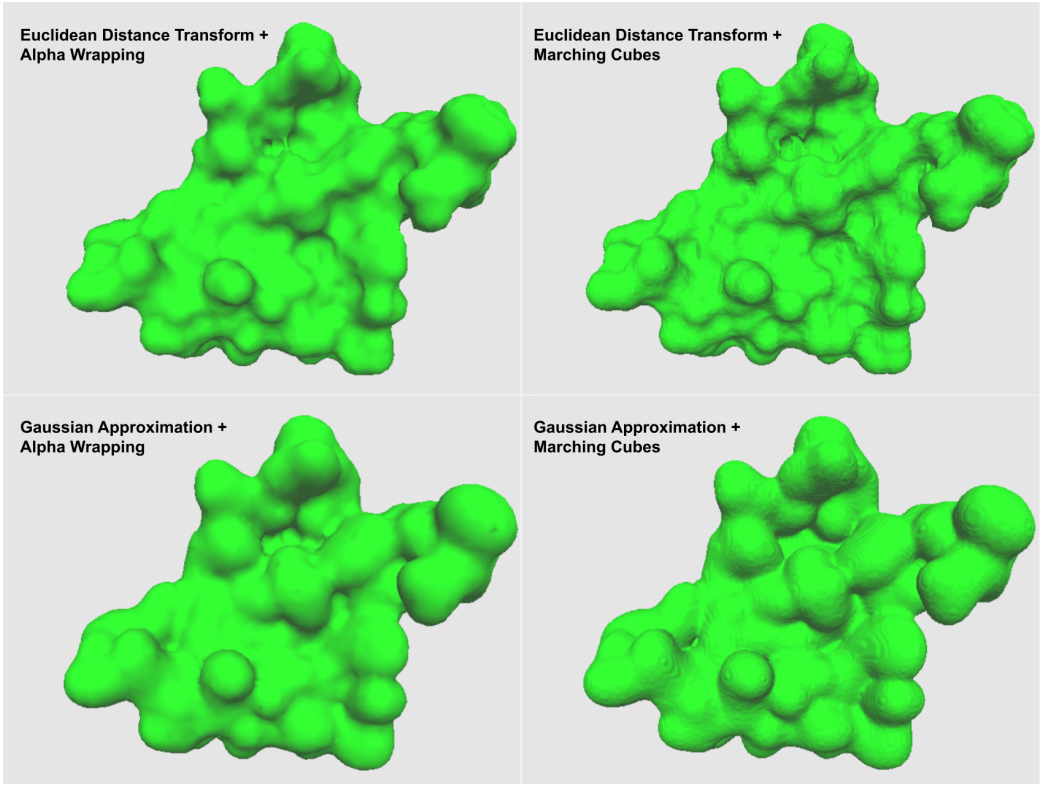

Figura 5.6: Confronto tra le combinazioni degli algoritmi disponibili

Figura 5.7: Viste differenti della molecola 2c7c nella demo grafica

Conclusioni

In questa tesi è stata sviluppata la libreria *Algomole*, uno strumento per lo sviluppo e la sperimentazione di algoritmi di meshing per le superfici molecolari. Questa libreria offre agli sviluppatori un framework comune per la generazione di modelli 3D di molecole a partire dalle coordinate atomiche, incluso un meccanismo di validazione dei risultati.

Sviluppi futuri

Nel corso del futuro sviluppo di *AlgoMole*, sarà importante esplorare e implementare ulteriori algoritmi di meshing e massimizzare le prestazioni di quelli già supportati. A questo scopo, si intende creare una comunità di ricercatori e sviluppatori interessati alle superfici molecolari e che vogliano contribuire all'evoluzione di questo strumento.

Inoltre, al momento Windows è l'unico sistema operativo supportato e testato. Verrà espanso il supporto per ulteriori piattaforme con anche il supporto di servizi di compilazione automatica come CMake.

Infine, dal punto di vista strutturale del progetto, un possibile sviluppo consiste nell'aumentare la possibilità di personalizzazione della pipeline. Al momento è possibile specificare un solo algoritmo per fase, fatto che limita molto il programmatore nel controllo del risultato. Si potrebbe quindi lasciare la possibilità all'utente di eventualmente specificare, per un dato passo della pipeline, una successione di sotto-passi. Per esempio, se si voles-

sero applicare due algoritmi diversi nella fase di *post-processing*, al momento sarebbe necessario creare un'intera terza classe che al suo interno semplicemente richiama i due algoritmi desiderati.

Il codice sorgente per *AlgoMole* può essere trovato all'indirizzo https://github.com/rob-lepore/algomole, mentre l'applicazione con interfaccia grafica che utilizza *AlgoMole* può essere scaricata all'indirizzo https://github.com/rob-lepore/algomole-demo.

Bibliografia

[1] B. Lee and F.M. Richards. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379–IN4, 1971. ISSN 0022-2836. doi: https://doi.org/10.1016/0022-2836(71)90324-X.

[2] Tolga Can, Chao-I Chen, and Yuan-Fang Wang. Efficient molecular surface generation using level-set methods. *Journal of Molecular Graphics and Modelling*, 25(4):442–454, 2006. ISSN 1093-3263. doi: https://doi.org/10.1016/j.jmgm.2006.02.012.

[3] F. M. Richards. Areas, volumes, packing and protein structure. *Annual Review of Biophysics and Bioengineering*, 6:151–176, 1977. doi: 10.1146/annurev.bb.06.060177.001055.

[4] Anurag Kumar and Kam YJ Zhang. Advances in the development of shape similarity methods and their application in drug discovery. *Frontiers in Chemistry*, 6:315, 2018. doi: 10.3389/fchem.2018.00315.

[5] Michael L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221(4612):709–713, 1983. doi: 10.1126/science.6879170.

[6] Michel F. Sanner, Arthur J. Olson, and Jean-Claude Spehner. Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996. doi: https://doi.org/10.1002/(SICI)1097-0282(199603)38:3⟨305::AID-BIP4⟩3.0.CO;2-Y.

[7] Evgeni V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, 1995.

[8] Dong Xu and Yang Zhang. Generating triangulated macromolecular surfaces by euclidean distance transform. *PLoS ONE*, 4(12):e8140, 2009. doi: 10.1371/journal.pone.0008140.

[9] C.Wayne Niblack, Phillip B Gibbons, and David W Capson. Generating skeletons and centerlines from the distance transform. *CVGIP: Graphical Models and Image Processing*, 54(5):420–437, 1992. ISSN 1049-9652. doi: https://doi.org/10.1016/1049-9652(92)90026-T.

[10] D.M. Gavrila. Multi-feature hierarchical template matching using distance transforms. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, volume 1, pages 439–444 vol.1, 1998. doi: 10.1109/ICPR.1998.711175.

[11] Junli Li and Xiuying Wang. A fast 3d euclidean distance transformation. In *2013 6th International Congress on Image and Signal Processing (CISP)*, volume 2, pages 875–879, 2013. doi: 10.1109/CISP.2013.6745288.

[12] Mesh smoothing, 2023. URL `https://graphics.stanford.edu/courses/cs468-12-spring/LectureSlides/06_smoothing.pdf`.

[13] Herbert Bernstein and Paul Craig. Efficient molecular surface rendering by linear-time pseudo-gaussian approximation to lee-richards surfaces (pgalrs). *Journal of applied crystallography*, 43:356–361, 04 2010. doi: 10.1107/S0021889809054326.

[14] G Grassmann, M Miotto, L Di Rienzo, G Gosti, G Ruocco, and E Milanetti. A novel computational strategy for defining the minimal protein molecular surface representation. *PLoS ONE*, 17(4):e0266004, 2022. doi: 10.1371/journal.pone.0266004.

[15] Daisuke Kihara, Lee Sael, Rayan Chikhi, and Juan Esquivel-Rodriguez. Molecular surface representation using 3d zernike descriptors for protein shape comparison and docking. *Current Protein and Peptide Science*, 12(6):520–530, 2011.

[16] Ryan Waite. What programming language is windows written in? Forum, 2009. URL `https://social.microsoft.com/Forums/en-US/65a1fe05-9c1d-48bf-bd40-148e6b3da9f1/what-programming-language-is-windows-written-in?forum=windowshpcacademic`.

[17] Programming with c++, 2023. URL `https://docs.unrealengine.com/5.1/en-US/programming-with-cplusplus-in-unreal-engine/`.

[18] Project chrono, 2023. URL `https://projectchrono.org/`.

[19] Sergio Polini. Introduzione alla programmazione orientata all'oggetto - prima parte. *MCmicrocomputer*, 1990.

[20] Eigen, 2023. URL `https://eigen.tuxfamily.org/index.php?title=Main_Page`.

[21] Cgal, 2023. URL `https://www.cgal.org/`.

[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns, Volume 1*. Wiley, 2000.