POLITECNICO DI TORINO

Corso di Laurea in Ingegneria del cinema e dei mezzi di comunicazione

Tesi di Laurea Magistrale

Generazione di istruzioni di cucitura elicoidali su una superficie parametrica

Relatori
Prof. Fabrizio Lamberti
Dr. Alberto Cannavò

Candidato
Massimo Gismondi
matricola: 270123

Sessione di aprile 2022

"Sharing knowledge is the most fundamental act of friendship.
Because it is a way you can give something without losing something."
Richard Stallman

Abstract

Negli ultimi decenni, grazie all'interesse dell'industria da un lato e dell'artigianato dall'altro, ci sono stati numerosi sforzi in direzione del *virtual prototyping* del settore tessile, ossia della digitalizzazione della fase di prototipazione, puntando a definire la superficie tridimensionale desiderata al calcolatore e delegando ad esso la generazione delle istruzioni di cucitura da seguire. Ciò è proficuo in quanto le relazioni tra la superficie *target*, la disposizione, il numero e la tipologia delle maglie da cucire, nonché le dimensioni dell'oggetto finito e la curvatura superficiale, non sono di immediata intuizione. Le istruzioni generate sono poi utilizzabili manualmente da un artigiano o possono essere convertite in comandi macchina per telai automatizzati.

Questa tesi adopera come riferimento principale l'algoritmo presentato da Çapunaman et al. [1] per affinità di obiettivo in termini di tecnica di cucitura, utilizzo di una struttura elicoidale ed impiego di superfici parametriche. A partire da questo lavoro viene proposto un nuovo algoritmo incrementale che, tramite la generazione di *short row* cucibili, ottimizzi la corrispondenza tra la superficie desiderata e quella dell'oggetto cucito riducendo le distorsioni delle maglie e che, al contempo, migliori le prestazioni dei tempi di calcolo, includendo inoltre una funzionalità di calcolo delle istruzioni e una di visualizzazione di anteprima.

Al fine di valutare l'efficacia di quanto realizzato, si comparano la distribuzione delle lunghezze e deformazioni delle maglie generate rispetto al caso ideale, i tempi di calcolo per quanto concerne il profilo quantitativo, e l'aspetto del risultato finale oltre alla sua regolarità per quello qualitativo.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

La tesi presenta un algoritmo incrementale per la generazione di istruzioni di cucitura, confrontato con la ricerca di Çapunaman et al. del 2017 [1] per affinità di obiettivo in termini di tecnica di cucitura, utilizzo di una struttura elicoidale fedele a quella reale ed impiego di superfici parametriche.

Essa è organizzata in quattro capitoli:

- Il primo capitolo introduce i concetti base di cucito necessari alla comprensione della ricerca e analizza gli approcci esistenti dell'attuale stato dell'arte, confrontandone ambito di applicazione, vantaggi e limitazioni.
- Il secondo capitolo tratta le primitive parametriche, curve e superfici, utilizzate per la definizione della superficie *target*. Si riassumono le ipotesi matematiche necessarie e si propone una riparametrizzazione che agevoli lo spostamento sulla superficie specifico per l'ambito di interesse.
- Il terzo capitolo concerne la realizzazione dello strumento software; si enuclea la struttura dati utilizzata, l'organizzazione dei file di progetto, si espone la scelta delle dipendenze software e si mostrano le differenze di approccio tra i due algoritmi in esame. In conclusione, si procede all'esposizione del metodo per la generazione delle istruzioni testuali e per la visualizzazione, comune a entrambi.
- Infine, negli ultimi due capitoli si confrontano i risultati ottenuti e si propongono migliorie per sviluppi futuri.

Si ringrazia Özgüç Çapunaman, uno degli autori del lavoro di riferimento, per aver messo a disposizione le superficie di esempio utilizzate nella sua ricerca, consentendo un più agevole confronto dei risultati.

Essendo il settore tessile molto ampio, si delinea l'ambito di ricerca di questa tesi. La realizzazione di un programma di *virtual prototyping* in ambito tessile obbliga a porre l'attenzione a tutti i punti enucleati nella Figura 1.1. Questa ricerca tocca tutti i punti inclusi nel riquadro tratteggiato, escludendo i macchinari industriali, per indisponibilità degli stessi, e lo sviluppo dell'interfaccia utente, poiché meritevole di un impegno di ricerca a sé stante. Tra i punti inclusi, il focus e le metriche sono improntate a valutare il "Calcolo e disposizione delle maglie", evidenziato in rosso.

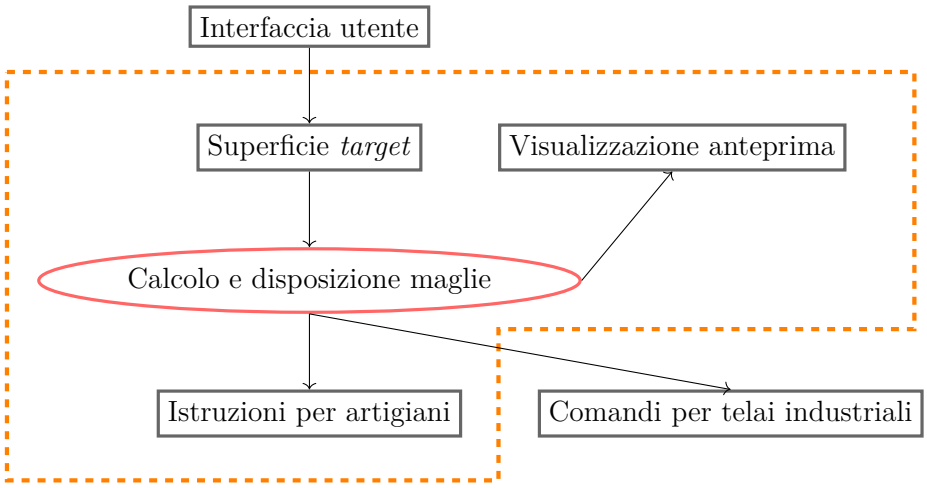


Figura 1.1: Inquadramento dell'ambito di ricerca.

I software per la generazione delle istruzioni di cucitura sono di interesse da un lato per gli hobbisti, dall'altro per le industrie tessili.

Prototipare rapidamente la forma e visualizzarla al calcolatore permette di risparmiare iterazioni di cucitura. La *pipeline* per la produzione di un oggetto finito solitamente consiste in:

- modellazione e progettazione oggetto;
- assegnazione parametri del cucito;
- eventuale simulazione fisica per *yarn relaxation*, ossia la predizione dell'assestamento della struttura cucita in base alle forze esistenti tra le singole maglie (il calcolo richiede da qualche secondo a qualche minuto);
- cucitura (ore o giorni).

Tra le varie fasi, la più esosa in termini di tempo è l'ultima, specialmente quando realizzata a mano. È di interesse spostare lo sforzo creativo in digitale, per ridurre la durata dei cicli di prototipazione e, di conseguenza, limitare il tempo totale investito nelle ultime due fasi.

1.1 A chi si rivolge

Come introdotto precedentemente, esistono due principali tipi di utenza: hobbisti e industrie tessili. Essi utilizzano un *workflow* comune per quanto concerne la parte iniziale di progettazione, la prototipazione al calcolatore e la simulazione, per poi divergere nell'ultima fase, ossia la generazione delle istruzioni vere e proprie.

1.1.1 Industria tessile

All'interno delle industrie tessili si usano telai automatizzati per il lavoro a maglia. I *designer* adoperano software per la definizione delle maglie. Essi collocano una maglia alla volta, aiutati dal software per creare *pattern* noti, per esempio per modificare l'aspetto del cucito. Il software poi compila le istruzioni per i macchinari.

Un macchinario industriale può ricevere delle istruzioni macchina, metaforicamente simile al g-code[1] delle stampanti 3D, che ne comandino i movimenti passo-passo. Nel caso di telai automatizzati, invece, vengono inviate informazioni su come spostare ogni ago, quale filo utilizzare e con quale tensione, oppure istruzioni speciali, ad esempio per la rotazione del cucito sul piano di lavoro. Anche il workflow è simile: modellazione → inserimento parametri della lavorazione → calcolo percorsi macchina → produzione automatizzata.

Un macchinario di questo tipo è veloce e non richiede intervento umano, ma ha alcune limitazioni di cui tenere conto nell'algoritmo:

- è limitato a cucire lungo la sola direzione del telaio;
- è ottimizzato per la lavorazione a maglia, ma non per altre tecniche come l'uncinetto;
- il numero di aghi del telaio è limitato e l'algoritmo deve gestire quelli disponibili, evitando di accumulare due volte il filo su aghi già adoperati nelle istruzioni precedenti;
- l'algoritmo può aver bisogno di fare ruotare il cucito sul telaio per avvicinare due parti; ciò richiede la generazione di istruzioni dedicate per il macchinario;
- maglie su aghi troppo distanti non possono essere unite senza rischiare di spezzare il filo.

1.1.2 Hobbistica e artigianato

Dall'altro lato, ha continuato a svilupparsi un filone di appassionati di cucito e artigiani. La lavorazione completamente manuale è in parte dovuta al costo proibitivo dei macchinari industriali.

Per quanto concerne il telaio, è possibile costruire un macchinario amatoriale seguendo le istruzioni del progetto OpenKnit [2], gestito da una scheda Arduino e con pezzi prodotti tramite stampa 3D. Il costo complessivo dei materiali, manodopera esclusa, si aggira intorno ai 380€ [3]. Da qualche anno l'ex-autore di OpenKnit ha lanciato un prototipo su Kickstarter [4], che punta a produrre un piccolo telaio automatizzato a uso casalingo. Tuttavia, non è molto economico, intorno ai 14000€, e ha scarsa flessibilità nel suo controllo. Infatti, il software è completamente proprietario[2], e l'utente è limitato all'uso della

[1] Il g-code è un linguaggio per le macchine a controllo numerico, o CNC. Istruisce un macchinario su operazioni e spostamenti da attuare sul piano di lavoro. Per esempio, può indicare a una fresatrice di spostare la fresa alle coordinate x: 50, y: 20, z: 2

[2] Un software è detto *con licenza proprietaria* quando la licenza scelta dall'autore non garantisce all'utente alcune libertà sul suo utilizzo, modifica e pubblicazione. L'opposto di un software proprietario è detto *software libero*, il quale garantisce le quattro libertà fondamentali https://www.gnu.org/philosophy/free-sw.html

web app fornita dal produttore. Se da un lato avere modelli pre-esistenti da modificare può semplificare l'uso per un utilizzatore base del macchinario, questo impedisce a strutture quali i Fablab[3], i *makerspace* in generale, artigiani e artisti di avere completa scelta creativa e personalizzazione, nonché limita l'integrazione di strumenti software esterni.

La cucitura manuale dà maggiore libertà di movimento e di creazione, a fronte di una ridotta velocità nella realizzazione di ogni singola maglia. Alcune maglie, per esempio quelle che richiedono un terzo ago di supporto o in cui bisogna congiungere punti molto distanti, sono realizzabili solo a mano. Nei telai è possibile congiungere solo maglie contigue sul letto del telaio. In generale, si riscontrano meno vincoli di comportamento dell'algoritmo.

Per l'approccio manuale, l'*output* non sarà più una sequenza di istruzioni macchina per gli attuatori del telaio, bensì un elenco di istruzioni testuali facilmente comprensibili. L'uomo beneficia anche di istruzioni visuali; per questo motivo, alcuni progetti hanno sviluppato un assistente visuale per rendere più chiaro il procedimento da seguire [5].

1.2 Tecniche di cucitura

I lavori di ricerca esistenti si concentrano principalmente su due tecniche di cucitura:

- a uncinetto, o "crocheting";
- a maglia, o "knitting".

La prima si mette in opera tramite un "uncinetto", appunto; esso è costituito da una singola barra di metallo, solitamente in alluminio, con una estremità uncinata. Dopo che ogni singola maglia è stata creata, sull'uncinetto rimane solo un anello singolo. Nella seconda, invece, si adoperano due aghi lunghi su cui vengono accumulate le maglie di ogni giro. Gli strumenti da adoperare per le due tecniche sono mostrati in Figura 1.2.

Ciò, oltre che per l'aspetto puramente pratico delle differenze nel metodo, le due tecniche danno vita a differenze estetiche del tessuto.

Si introducono di seguito alcuni fondamentali del cucito, necessari per la comprensione dell'algoritmo.

1.2.1 Dimensione filo e maglie

Ogni oggetto può essere costruito con maglie di dimensione diversa, come illustrato nella Figura 1.3. Ciò influenza sia l'aspetto, sia i tempi di lavorazione. La grandezza della maglia può essere variata adoperando un filo di diametro diverso.

Tuttavia, la dimensione effettiva non dipende univocamente dal diametro del filo, bensì anche dalla tensione applicata e dalle caratteristiche stilistiche della "mano" di ogni artigiano.

Allo scopo di tenere in considerazione anche l'ultima variabile, Çapunaman suggerisce di far cucire all'artigiano una griglia semplice 10×10, misurarne larghezza e altezza media di ogni maglia e impostare nello strumento software i valori da lì ricavati.

[3] I Fablab sono una rete di laboratori condivisi di artigianato digitale. Promuovono ricerca sul territorio e tra gli obiettivi hanno anche il cucito 2.0.

(a)

(b)

Figura 1.2: Due tecniche di cucitura artigianale: (a) Esempio di cucitura all'uncinetto. Foto di Castorly Stock da Pexels; (b) Strumenti per il lavoro a maglia. Foto di Miriam Alonso da Pexels.

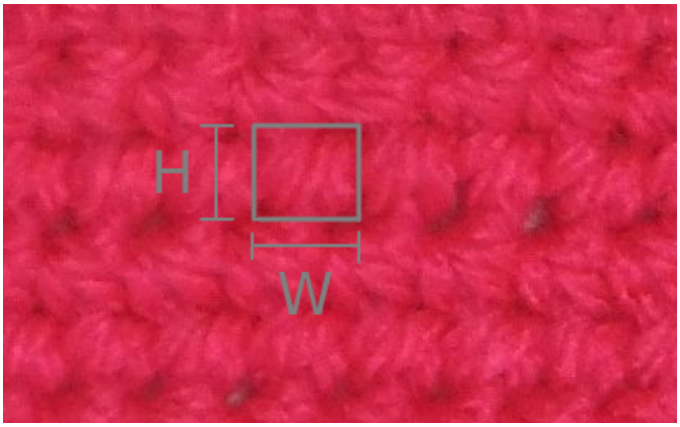


Figura 1.3: Estrarre la larghezza e l'altezza medie di una maglia.

Poiché questa tesi espande quella ricerca, è stato mantenuto il supporto a questa caratteristica, con i valori impostabili globalmente nella fase precedente alla definizione della

superficie.

1.2.2 Aumenti e diminuzioni

Si immagini, per semplicità, di voler realizzare un semplice cilindro. Come primo passo, si realizza il cerchio di base del diametro desiderato, ottenibile adoperando la larghezza media delle maglie precedentemente impostata. Lo scopo è creare nuove maglie per estrudere il cerchio in altezza, al contempo mantenendo lo stesso diametro: a ogni maglia del livello precedente sarà collegata una sola maglia del livello successivo. In questo modo, il diametro rimane costante e l'oggetto realizzato approssima un cilindro. La maglia standard corrispondente è detta *maglia bassa*. È abbreviata come *mb* nella notazione italiana e *sc* (*single crochet*) in inglese.

Si ipotizzi, invece, di voler realizzare un tronco di cono. È possibile incominciare con una circonferenza di base del diametro corretto, ma è necessario considerare il cambiamento del diametro nel tempo man mano che si procede con la cucitura. Ad esempio, si potrebbe decidere di passare da 29 a 32 maglie, in modo da allargare il diametro tra un livello e il successivo. Ad alcune delle maglie del livello precedente saranno collegate due di quello successivo. L'operazione corrispondente è detta *aumento*. È abbreviata come *aum* nella notazione italiana e *inc* (*increase*) in inglese.

Al contrario, per ridurre il diametro nel livello successivo, per esempio passando da 32 a 29 maglie, sarà necessario collegare una coppia di maglie del livello precedente a una sola di quello successivo. L'operazione corrispondente è detta *diminuzione*. È abbreviata come *dim* nella notazione italiana e *dec* (*decrease*) in inglese.

Riassumendo, l'algoritmo può inserire i tre tipi di collegamenti tra livelli:

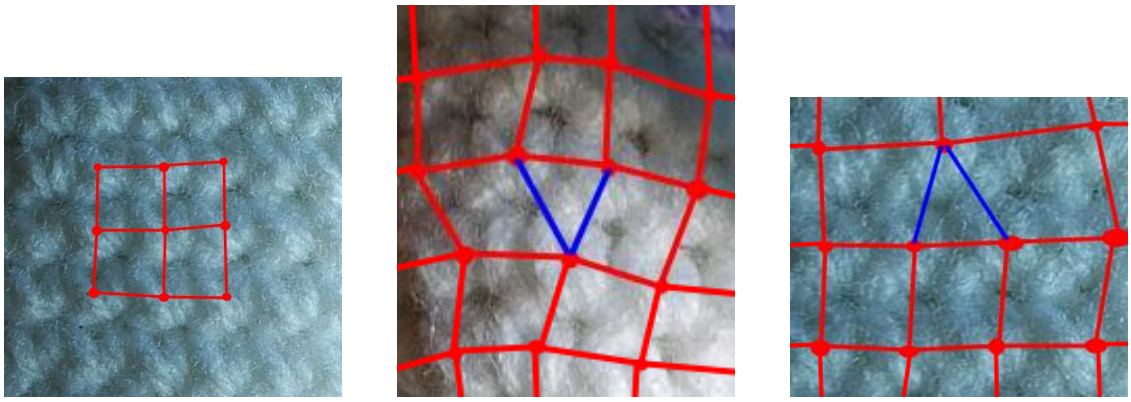
- maglia bassa, collegamento uno a uno;
- aumento, collegamento uno a due;
- diminuzione, collegamento due a uno.

Anche nel cucito risultante, mostrato in Figura 1.4, si può distinguere l'andamento di questi tre tipi di maglie localmente, oltre che nella variazione del diametro dell'oggetto.

1.2.3 Short row

Avendo a disposizione solo aumenti e diminuzioni, ci sarebbero forti limitazioni nelle forme realizzabili. Sarebbe attuabile creare solo forme con simmetria cilindrica, quindi cilindri o sfere, il cui diametro non varî in modo eccessivo, pena una approssimazione inaccurata della superficie.

Poiché l'altezza di ogni maglia è pressoché costante e il filo di lana non ha grande elasticità, si deve prestare attenzione a non applicare grandi forze di trazione e compressione, a non spezzare il filo e a non creare stiramenti o rigonfiamenti della superficie che risulterebbero esteticamente sgradevoli. Mantenere la distanza di ogni coppia di nodi da congiungere ad un valore simile alla lunghezza naturale delle maglie è un obiettivo primario per puntare a una buona approssimazione della superficie e una buona resa estetica del risultato. Qualora si tenti di curvare una forma, per esempio deformando un cilindro lungo



(a) (b) (c)

Figura 1.4: Differenze visive tra maglie semplici, aumenti e diminuzioni. (a) Una normale maglia bassa, senza aumenti né diminuzioni; (b) In blu, un aumento; (c) In blu, una diminuzione.

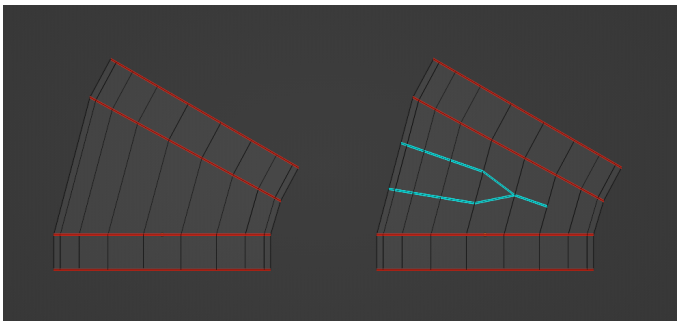


Figura 1.5: Effetto della compensazione tramite short row

una curva e perdendo la simmetria cilindrica, si deve intervenire per compensare l'altezza sul lato esterno della curva.

Di base, la cucitura avviene come un'unica spirale, costruendo sull'intero giro del livello precedente e mantenendo la direzione di cucitura costante; tuttavia, è possibile aggiungere dei livelli parziali, lunghi una frazione dell'intero giro, allo scopo di aggiungere contributi di crescita della superficie in modo asimmetrico. Tali livelli parziali sono detti *short row*. Nella Figura 1.5 è presente un cilindro curvato. In rosso sono evidenziati i livelli circolari che si desidera collegare. In azzurro le short row aggiunte come compensazione della distanza tra essi. In questo modo, l'altezza di ogni maglia è mantenuta ad un valore simile a quello originario.

Al contrario, nell'oggetto sul lato destro non è stata inserita alcuna short row. La lunghezza dei lati sul lato sinistro della superficie destra, ossia delle maglie, è circa tre volte il loro valore originario. Una struttura come questa, una volta cucita, sarebbe un cilindro perfettamente verticale anziché curvato. Se si tentasse di forzare la curvatura, si rischierebbe di rompere il filo sul lato esterno della curva oppure di creare deformazioni antiestetiche sul lato interno.

(a) (b)

Figura 1.6: I due approcci al calcolo delle maglie; (a) approccio di calcolo ad anelli sovrapposti; (b) approccio di calcolo elicoidale.

1.2.4 Cucitura elicoidale

Esiste una tecnica di cucitura elicoidale, solitamente utilizzata per gli *amigurumi*[4] all'uncinetto.

In Figura 1.6 è mostrato un cilindro percorso con due metodi diversi, ossia ad anelli sovrapposti oppure in modo elicoidale. La maggior parte degli algoritmi esistenti usa un metodo di calcolo ad anelli sovrapposti, nonostante la cucitura sia in realtà prodotta in modo elicoidale. La tecnica dell'uncinetto consente di produrre strutture con entrambi gli approcci: tuttavia, quello ad anelli sovrapposti renderebbe necessario inserire una maglia di aspetto diverso in corrispondenza dei salti tra un anello e il successivo oppure esporre il lato interno delle maglie all'esterno, causando una giustapposizione estetica evidente. A livello puramente strutturale funzionerebbe correttamente, ma peggiorerebbe la resa estetica. Lavorando in modo elicoidale si ottiene maggiore uniformità dell'aspetto della superficie: la direzione di cucitura viene mantenuta, non sono presenti maglie di salto tra i livelli e ogni maglia ha sempre il proprio lato esterno rivolto verso l'esterno dell'oggetto.

Inoltre, poiché il risultato di questa tesi può essere usato in software di prototipazione, nei quali il ruolo dell'anteprima non è mostrare una immagine solo esteticamente piacevole, bensì che sia più fedele possibile al risultato finale, l'algoritmo progettato opera in modo elicoidale *by design*.

Come introdotto precedentemente, gli approcci della ricerca in questo campo si suddividono per tecnica di cucitura, ambito industriale o artigianale, modellazione parametrica o poligonale.

Alcuni lavori incorporano tutte le fasi del workflow, altri si limitano a espanderne una sola.

[4]Nella cultura del Giappone, gli amigurumi sono piccoli pupazzetti imbottiti interamente lavorati all'uncinetto.

1.3 Fase di design della superficie

È necessario un metodo di definizione di una superficie che guidi gli algoritmi alle fasi successive.

1.3.1 Poligonale

È molto diffuso l'uso di una *mesh* poligonale come input. Tra questi, alcuni operano un *remesh* preliminare, allo scopo di ridisporre le maglie con la densità, direzione e numero di lati che si aspetta l'algoritmo alla fase successiva [6]. La superficie viene campionata con degli *iso-contours* tra due bordi, a loro volta campionati in base alla larghezza della maglia. Il risultato è una mesh principalmente formata da quadrilateri, con eccezionalmente alcuni triangoli.
Altri necessitano di una mesh unicamente con facce triangolari [7]; l'operazione di remesh uniforma l'area di ogni triangolo, ma senza introdurre quadrilateri. Se è vero che usare come base un modello 3D può essere utile per realizzare un software generico che, per esempio, generi le istruzioni ricevendo un qualsiasi file in formato `.stl`[5] (in rete sono disponibili molti modelli [8] con licenze Creative Commons[6]), molti modelli sono in ogni caso eccessivamente dettagliati per poter essere cuciti.

La mesh può essere realizzata tramite programmi esterni di modellazione, come Blender, Sketchup, OpenSCAD, oppure tramite interfacce semplificate. In questa ultima categoria si collocano progetti come il software Knitty [10], realizzato a scopo di ricerca nel 2008. Il suo scopo è simile a quello di questa tesi, ossia semplificare la prototipazione. L'utente può tracciare delle linee come in un programma di disegno 2D. Per esempio, nel caso l'utente tracci un rettangolo, questo verrà interpretato come un cilindro con diametro pari al lato minore del rettangolo. La normale della base del cilindro verrà allineata con la normale della superficie principale a cui si desidera agganciarlo. In conclusione, permette di generare un modello tridimensionale intermedio da degli *sketch* semplici, poi convertito in istruzioni di cucitura. È stato utilizzato in un workshop dell'autore con ragazzi tra i 10 e i 14 anni, per valutarne l'intuitività.

Un approccio ibrido, non considerabile però come totalmente parametrico, è l'uso di OpenSCAD. Esso permette di definire modelli 3D tramite un linguaggio di programmazione. Nonostante sia possibile modificare l'output in funzione dei parametri e delle variabili inseriti nel codice, l'output è un file in formato `.stl`, dunque una mesh con facce triangolari.

1.3.2 Parametrica

La seconda strategia è definire la superficie in maniera completamente parametrica. Come vantaggio, si può tenere conto delle variabili usate nella definizione della superficie per

[5] Il formato `.stl` consente di salvare modelli 3D. Supportando solo facce triangolari, è usato come formato di interscambio quando non sia necessario mantenere la topologia, per esempio nel campo della stampa 3D

[6] Le licenze creative commons proteggono le opere creative promuovendone il riuso. Le condizioni dipendono dalla licenza specifica scelta dall'autore dell'opera creativa [9]

agevolare l'algoritmo nelle fasi successive. A scopo di esempio, la modellazione parametrica mantiene informazioni sulla direzione di sviluppo della superficie, che potrebbe essere utilizzata come direzione di cucitura; tuttavia, se essa venisse esportata come mesh, queste informazioni verrebbero perdute. Molti degli algoritmi basati su mesh richiedono un intervento manuale dell'utente per ricostruire i dati mancanti, per esempio tramite un etichettamento manuale o semiautomatico delle facce [5] oppure tramite la definizione di un campo scalare in $\mathbb{R}^3$ tramite l'assegnazione di pesi ai vertici per guidarne il comportamento [11].

Alcuni esperimenti, come ad esempio il lavoro di Osinga et al. [12], adoperano espressioni analitiche arbitrariamente complesse, tra cui l'attrattore di Lorentz, superficie definita da un sistema di tre equazioni differenziali. La generazione delle istruzioni inizia da un cerchio in una zona stabile, che viene espanso iterativamente a una distanza pari all'altezza di una maglia. La mesh così calcolata è stata convertita in istruzioni di cucitura, poi eseguite manualmente. La struttura è sorretta da filo di ferro opportunamente sagomato. Nonostante lo studio non miri a un utilizzo artigianale o industriale pratico, mostra le potenzialità della cucitura, in questo caso all'uncinetto, per esprimere forme molto complesse.

In ultimo, sono adoperate superfici con parametrizzazione UV, nelle quali la direzione di cucitura coincide con quella di U [13]. La modellazione parametrica mostra dei limiti di interoperabilità. Al contrario dei numerosi formati di interscambio esistenti per le mesh, quali `.stl` e `.obj`, nelle superfici parametriche la scelta del software in cui essa è definita diventa un limite. Ad esempio, uno degli algoritmi proposti [1] opera con la funzione NURBS UV in Grasshopper all'interno di Rhinoceros 3D. Altri software potrebbero avere funzionalità parametriche diverse implementate al loro interno.

1.4 Fase di calcolo delle maglie

L'obiettivo di ogni algoritmo è calcolare una struttura a rete che approssimi la superficie originale e che eviti eccessive forze di trazione o compressione tra le maglie.

Vengono utilizzati tre approcci concorrenti:

- approccio globale;
- approccio incrementale;
- compilatori.

Della prima categoria fanno parte lavori come quello di Wu [14], in cui viene proposto un metodo completamente automatizzato per convertire forme poligonali arbitrarie in una *stitch mesh*[7]. Notando che nel caso alcune maglie non siano collegate a quelle successive esse si disfarrebbero, vengono corrette a posteriori in maniera automatica. L'algoritmo usa una pipeline a parametrizzazione globale, seguita da alcuni operatori topologici per le direzioni di cucitura e da una fase di ottimizzazione, sempre globale, per ridurre le irregolarità. Una limitazione è rappresentata dal fatto che, nonostante la struttura sia

[7] La rappresentazione in stitch mesh serve per visualizzare la struttura risultante mantenendo la relazione tra le singole maglie

valida da un punto di vista degli intrecci del filo, motivo per cui è adatta ad effettuare simulazioni fisiche, essa non è cucibile su un telaio automatizzato. Per questo motivo, anche le immagini presenti nel lavoro sono state generate in computer grafica oppure in plastica tramite una stampante 3D.

Altri lavori adoperano un approccio incrementale: l'algoritmo calcola le informazioni di cucitura spostandosi sulla superficie un livello alla volta, anziché globalmente. Impostato un campo scalare e un vertice di partenza, l'algoritmo procede passo passo a generare un grafo, guidato dal campo [11].

Il lavoro di questa tesi appartiene alla categoria incrementale.

Dell'ultima categoria fanno parte algoritmi non basati su superfici, bensì dei compilatori [15] di primitive interne al programma in istruzioni macchina per i telai.

1.4.1 Short row

Il problema delle short row, introdotto nella Sezione 1.2.3, è considerato solo da ricerche recenti. Il sopracitato compilatore [15] consente l'inserimento delle short row come primitiva a sé stante, dunque alternando manualmente cilindri a sezioni contigue di short row, che risulta in una piega improvvisa della direzione e uno spigolo aspro sul lato esterno. Ciò è adatto alla produzione di vestiti, ma riscontra dei limiti quando applicato ad oggetti più complessi.

Altri algoritmi inseriscono short row in maniera più graduale autonomamente, alternate alle maglie dei giri normali [13, 11]. Ciò consente di realizzare curve morbide.

1.4.2 Eliche

Considerare la struttura ad anelli sovrapposti semplifica l'algoritmo, ma è meno vicina alla struttura reale. Ciò può essere ignorato quando l'altezza della maglia sia trascurabile rispetto alle dimensioni dell'oggetto, per esempio degli abiti. Il tema viene affrontato in un lavoro molto recente [5], in cui si propone un'estensione delle stitch mesh che consenta di rappresentare strutture a spirale.

In questa tesi si punta a mantenere una struttura più simile a quella reale, dunque elicoidale per motivi di aderenza dell'anteprima all'oggetto fisico completato.

1.5 Migliorie

Riassumendo quanto esposto finora e facendo riferimento ai lavori di ricerca trovati, come indicato nella Tabella 1.1 non esistono algoritmi per alcuni casi d'uso. In questa tesi si presenta un algoritmo che sia contemporaneamente:

- basato su una superficie parametrica rendendo l'algoritmo più accessibile, in quanto non richiede conoscenze di modellazione 3D o sculpting;
- abbia tempi di calcolo adatti a un uso interattivo o semi-interattivo;
- consideri l'aggiunta automatica di short row per curvature graduali;

	Supporto short row	Poligonale	Parametrica
Anelli sovrapposti	No	✓ [6]	✓ [12]
	Sì	✓ [11]	✓ [15, 13]
Elicoidale	No	X	✓ [1]
	Sì	(parzialmente) [5]	X

Tabella 1.1: Confronto delle funzionalità

- faccia ricorso a una struttura elicoidale, in modo da ottenere un'anteprima più coerente possibile con il prodotto finito dal punto di vista estetico e strutturale.

A fronte di quanto illustrato, l'algoritmo proposto mantiene sia l'uso di superfici parametriche, sia la struttura elicoidale; come detto la tesi adopera come riferimento la ricerca di Çapunaman [1] per le affinità nella loro categorizzazione e scopo. Viene integrato il supporto alle short row nella struttura, una generazione di istruzioni e una parte di visualizzazione e rendering.

Capitolo 2

Primitive modellazione parametrica

Poiché le superfici valide nel cucito, nonché quelle fissate dal lavoro di riferimento, sono esprimibili come dei cilindri generalizzati parametrizzabili UV, in questa tesi esse sono definite come tali. Le superfici di interesse successivamente analizzate vengono determinate specificando alcuni parametri quali il raggio punto per punto, la curva di base da estrudere e la curva di controllo su cui deformarla.

Si espongono qui i concetti di algebra adoperati per la definizione delle curve e della superficie stessa, nonché le proprietà necessarie per il funzionamento dell'algoritmo.

2.1 Curve

Una curva nello spazio 3D è descrivibile come una funzione parametrizzata da una singola variabile, che associa un intervallo I in $\mathbb{R}$ a valori in uno spazio con numero arbitrario di dimensioni. Ai fini di questa ricerca, I è un intervallo chiuso su $\mathbb{R}$ come $[0, 1]$ e X lo spazio di arrivo $\mathbb{R}^3$.

$$f : I \mapsto X$$

$$f : [0, 1] \mapsto \mathbb{R}^3$$

2.1.1 Definizione delle curve

Classi C^n

È utile ricordare il concetto di *classe* di una funzione, in questo caso applicato a una curva, in base alla sua derivabilità con continuità. Una funzione è di classe C^0 se non ha derivata continua su tutto il dominio originale. Una funzione è di classe C^1 se la derivata prima è continua. Di conseguenza, una funzione di classe C^n sarà derivabile con continuità n volte. Per estensione, una funzione di classe C^∞ sarà derivabile con continuità un numero infinito di volte.

Parametrizzazione intrinseca

Non sempre la parametrizzazione su t è utile. Nel calcolo del raggio della circonferenza osculatrice o quando sia necessario stimare le distanze sul sostegno della curva, è comodo adoperare la lunghezza d'arco. Quando la curva è parametrizzata tramite la lunghezza d'arco anziché tramite t, si dice che è espressa in "parametrizzazione intrinseca" [16].

Data una curva $f(t)$, si definisce una funzione $s(t)$:

$$s(t) = \int_{t_0}^{t} |f'(u)|du$$

Poiché saranno utilizzate curve parametrizzate con $t \in [0, 1]$, si sostituisce $t_0 = 0$. Si richiede siano rispettate le condizioni seguenti:

- $f(t)$ sia almeno di classe C^1;
- $|f'(t)| \neq 0, \forall t \in I$.

Il primo requisito assicura che anche $f'(t)$ è continua. Dal momento che è una funzione continua, è sicuramente integrabile. La seconda condizione garantisce che il verso di percorrenza della curva non si inverta e che non si aggiungano contributi pari a zero durante l'integrazione. Da ciò consegue che $s(t)$ è monotona strettamente crescente, condizione sufficiente affinché sia invertibile. Si definisce:

$$t(s) := s^{-1}(t)$$

La curva riparametrizzata tramite la lunghezza d'arco s sarà $f(t(s))$.

Si indicherà con $f(t)$ una curva parametrizzata tramite t, e come $f(s)$ una curva in parametrizzazione intrinseca.

2.1.2 Sistemi di riferimento

Si possono definire alcuni sistemi di riferimento lungo la curva, i cui versori dipendono dal parametro t. Tra questi, il *frame* di Frenet-Serret può essere definito nel caso di una curva biregolare, in cui $f'(t) \times f''(t) \neq 0$, ossia la derivata prima e la derivata seconda sono linearmente indipendenti e $|f''(t)| \neq 0$[1]. Necessitando della derivata seconda, $f(t)$ deve essere derivabile due volte. Per motivi estetici, si può richiedere che anche la derivata seconda sia continua, quindi che $f(t) \in C^2$.

Per quanto concerne la notazione, si denota con $|\vec{v}|$ l'operatore di calcolo della norma euclidea e con $norm(v)$ l'operazione di normalizzazione.

Nel frame di Frenet sono definiti tre versori:

- $\hat{T}(t)$ versore tangente, sempre parallelo alla tangente all'istante t;
- $\hat{N}(t)$ versore normale, diretto verso il centro della circonferenza osculatrice;

[1] Per quanto concerne la derivata prima, la condizione $f'(t) \neq 0$ era già stata espressa come ipotesi precedentemente.

- $\hat{B}(t)$ versore binormale, prodotto vettoriale dei primi due; ciò definisce il piano su cui giace la circonferenza osculatrice.

Il versore tangente $\hat{T}(t)$ è ottenibile semplicemente normalizzando la derivata prima:

$$\hat{T}(t) = norm(f'(t))$$

Il versore normale $\hat{N}(t)$, invece, ottenibile come:

$$\hat{N}(t) = norm(f''(t) - \langle f''(t), \hat{T}(t) \rangle \cdot \hat{T}(t))$$

Oppure, sfruttando la parametrizzazione intrinseca $t = t(s)$:

$$\hat{N}(t) = norm(f''(t(s)))$$

Infine,

$$\hat{B}(t) = \hat{T}(t) \times \hat{N}(t)$$

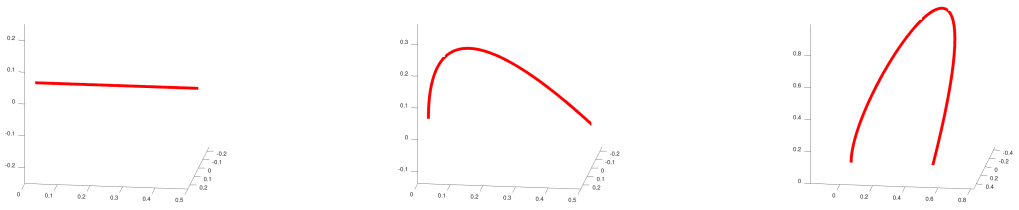
In alternativa, è possibile calcolare prima il versore binormale come $\hat{B}(t) = norm(f'(t) \times f''(t))$ e successivamente ricavare $\hat{N}(t) = -\hat{T}(t) \times \hat{B}(t)$.

Limitazioni

Il Frame di Frenet-Serret non è definito quando la curvatura si annulla. Inoltre, il verso del versore normale e binormale cambia improvvisamente in corrispondenza di un punto di flesso. Poiché nel caso d'uso considerato non è possibile escludere che ciò avvenga, sarà necessario espanderlo con un altro sistema di riferimento.

Ovviando al problema di discontinuità del Frame di Frenet, nella tesi si utilizza un sistema di riferimento più stabile $(\hat{i}(t), \hat{j}(t), \hat{k}(t))$ e definito su ogni punto della curva di controllo, con lo scopo di ignorare la torsione della curva. Viene costruita una LUT[2] con cui ricavare velocemente il versore $\hat{j}(t)$. Per valori intermedi tra quelli campionati dalla LUT, $\hat{j}(t)$ è ottenuto tramite una rotazione tra il valore campionato precedente e quello successivo esistenti nella LUT; questa operazione è rappresentata come una interpolazione tra due quaternioni. Poiché da una semplice interpolazione lineare risulterebbe una scalatura, il quaternione interpolato viene normalizzato. Nello strumento software, ciò è ottenuto con il metodo `nlerp()` della libreria `nalgebra`, come dettagliato in seguito alla Sezione 3.1.1. Tramite questo passaggio è stato ricavato velocemente $\hat{j}(t)$. Avendo definito $\hat{i}(t)$ e $\hat{j}(t)$ su tutto il dominio e poiché essi sono sempre ortonormali, si può poi calcolare $\hat{k}$ tramite semplice prodotto vettoriale $\hat{k}(t) = \hat{i}(t) \times \hat{j}(t)$, ottenendo un triedro di versori ortonormali.

[2] In generale, vengono utilizzate ogni qualvolta si desidera precalcolare una trasformazione; come esempio della loro applicabilità in ambiti disparati, sono usate anche nei filtri per color grading per l'elaborazione di immagine e video.



(a) (b) (c)

Figura 2.1: Sostegno di curve di Bézier definite da due, tre o quattro punti; (a) Bézier lineare, due punti estremi; (b) Bézier quadratica, due punti estremi e uno di controllo; (c) Bézier cubica, due punti estremi e due di controllo

2.1.3 Bézier

Allo scopo di questa tesi, la funzione viene definita tramite una curva di Bézier. Quelle più utilizzate prevedono di specificare da due a quattro punti. La curva passa attraverso il primo e l'ultimo punto. Invece, gli eventuali punti intermedi sono usati solo come *punti di controllo*. Lo strumento software sviluppato implementa sia curve lineari, quadratiche o cubiche, rispettivamente definite da due, tre e quattro punti. Le tre varianti sono riportate in Figura 2.1.

Nel caso, come esempio, di una curva cubica $f(t)$, essa è definita come

$$f(t) := (1-t)^3\vec{P_0} + 3(1-t)^2 t\vec{P_1} + 3(1-t)t^2\vec{P_2} + t^3\vec{P_3}, \text{ con } t \in [0,1]$$

La derivata prima e la derivata seconda possono essere calcolate analiticamente. Ottenute le derivate e verificato che la derivata prima e la derivata seconda siano linearmente indipendenti, esse possono essere inserite nelle formule precedenti per ottenere i versori del sistema di riferimento di Frenet-Serret.

LUT

Nonostante la curva di Bezier abbia una rappresentazione analitica spesso in computer grafica, per velocizzare le operazioni su queste primitive vengono usate delle LUT. Alla creazione della primitiva il software campiona l'intervallo con un numero arbitrario n di punti. I punti del sostegno calcolati vengono mantenuti in memoria per un accesso veloce. In seguito, quando si desidera ricavare $f(t)$ in un punto t arbitrario, il software trova l'indice nella LUT ad esso più vicino. Nel caso t non sia uno dei valori pre-calcolati, ma giaccia tra due valori campionati t_0 e t_1, è possibile calcolare un valore approssimato di $f(t)$ in maniera molto semplice interpolando $f(t_0)$ e $f(t_1)$, di cui era stato precalcolato il risultato.

2.2 Superfici

Una superficie parametrica è definita da una funzione a due variabili (u, v). In questa tesi, la variabile u è adoperata per spostarsi sulla curva di controllo $C(u)$. Si noti che il parametro della curva di controllo, prima chiamato generalmente t durante la trattazione

delle curve, per evitare ambiguità verrà denominato u quando riferito alle coordinate della superficie parametrica e sarà anch'esso compreso tra 0 e 1. A ogni $u \in [0, 1]$ è associabile una curva $Q(v)$. Combinando i due parametri (u, v), si ottiene una superficie.

In questa tesi, si definiscono N curve $Q_n(v)$, associate ai rispettivi u_n sulla curva di controllo. Dopodiché, per ottenere il valore della superficie in $(\overline{u}, \overline{v})$ desiderato, si trovano le curve $Q_n(v)$ associate tra cui il $\overline{u}$ cercato è compreso. Denominando $Q_k(v)$ la curva successiva al parametro scelto, si calcola il punto sulla superficie interpolando $Q_k(\overline{v})$ e $Q_{k-1}(\overline{v})$.

Per costruire le curve $Q_n(v)$ e disporle lungo la curva di controllo in modo da definire la superficie, viene utilizzato il $\hat{i}(u), \hat{j}(u), \hat{k}(u)$ esposto alla Sezione 2.1.2.

Riassumendo, si può definire una trasformazione sulla curva C che associ le coordinate UV (u, v) al punto sulla superficie, passando per il sistema di riferimento IJK:

$$(u, v) \mapsto (x, y, z)$$

Ogni punto della superficie può essere espresso con le due coordinate u, v, entrambe appartenenti all'intervallo [0,1].

$$\vec{P}(u, v) = \begin{bmatrix} i_x(u) & j_x(u) & k_x(u) \\ i_y(u) & j_y(u) & k_y(u) \\ i_z(u) & j_z(u) & k_z(u) \end{bmatrix} \begin{bmatrix} Q_x(v) \\ Q_y(v) \\ Q_z(v) \end{bmatrix} + \begin{bmatrix} C_x(u) \\ C_y(u) \\ C_z(u) \end{bmatrix}$$

I punti sulla curva Q, originariamente giacenti sul piano $z = 0$, vengono ruotati tramite una matrice di rotazione che ha come colonne i versori destinazione, e poi successivamente traslati di $\vec{C}(u)$.

Se si utilizzasse il frame di Frenet-Serret per definire i versori $\hat{i}(u), \hat{j}(u), \hat{k}(u)$, risulterebbero alcune discontinuità del verso e della direzione dei due versori ortogonali a $\hat{i}(u)$ nei punti di flesso della curva di controllo o, peggio, non sarebbero definiti nel caso in cui la curva di controllo abbia curvatura localmente nulla.

2.2.1 Riparametrizzazione

Come introdotto nella Sezione 1.4.2, molti algoritmi sono semplificati tramite il calcolo ad anelli sovrapposti anziché a spirale, associando ogni isocurva ad un livello di cucitura. Si introduce dapprima una rappresentazione ad anelli sovrapposti, per poi estenderla ad una elicoidale utile ai fini delle funzionalità dell'algoritmo presentato.

Si ricorda, come requisito conseguente dall'obiettivo della tesi, che la distanza minima percorribile cucendo è pari all'altezza di una maglia, quantità fissata a inizio lavorazione. È necessario assicurarsi che essa venga percorsa in ogni passo dell'algoritmo. Nella Figura 2.2 sono rappresentate delle circonferenze disposte affinché la distanza minima tra ogni coppia sia pari alla altezza della maglia impostata; eventuali distanze maggiori presenti tra esse potranno essere coperte tramite short row. La distanza minima tra le due circonferenze, nel caso di un cilindro a raggio variabile, si concretizza nel verso del versore normale della curva di controllo $\hat{N}(t)$, quella massima nel verso opposto $-\hat{N}(t)$. Nel caso la curva di controllo fosse un segmento, il grafico si ridurrebbe a un caso semplice di circonferenze equispaziate.

Si definisce una nuova parametrizzazione della superficie $(u, v) \mapsto (\tau, v)$ tale che l'estremo superiore di τ corrisponda al n-esimo anello. La coordinata u, inizialmente compresa

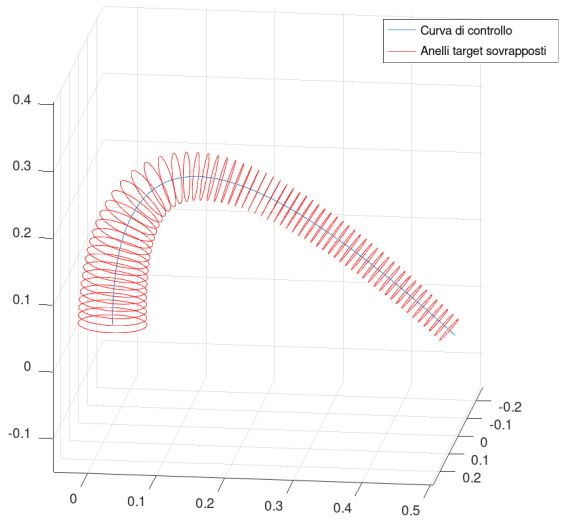


Figura 2.2: Anelli distanziati disposti lungo una curva.

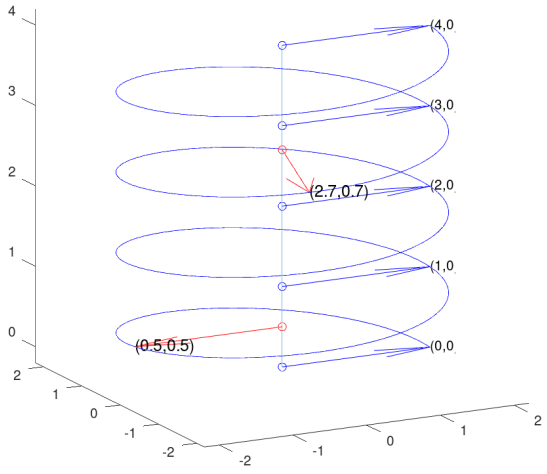


Figura 2.3: Punti dell'elica espressi nel sistema (τ, v).

tra 0 e 1 e utilizzata per definire la superficie $S(u, v)$, è stata riparametrizzata e ci si riferirà ad essa come $S(\tau, v)$. Esempi di punti espressi nella nuova parametrizzazione sono riportati in Figura 2.3.

Tramite la definizione di questa riparametrizzazione l'algoritmo, in qualsiasi punto della superficie si trovi, può facilmente conoscere quale sia il numero di maglie da realizzare per raggiungere il livello successivo mantenendo sempre, per costruzione, il requisito che la

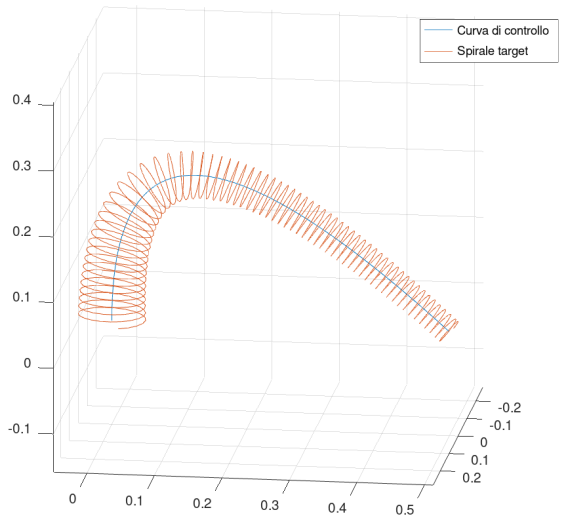


Figura 2.4: Spirale disposta lungo la curva

minima distanza tra livelli sia pari all'altezza di una maglia. Da un punto arbitrario (τ_0, v_0) si può aggiunge 1, ottenendo $(\tau_0 + 1, v_0)$; esso è il punto in cui passa la isocurva successiva. Dalla misurazione della distanza geodetica tra i due punti, si ricava il numero di maglie necessarie per congiungerli, che per costruzione è stato assicurato sia ≥ 1.

Si può costruire una spirale principale sulla superficie $S(\tau, v)$ in Figura 2.4, definibile come:

$$E(\tau) = S(\tau, mant(\tau))$$

Dove $mant(x)$ è la funzione mantissa[3]. Questa rappresentazione garantisce inoltre che, da qualsiasi coordinata (τ_0, v_0) si parta, anche per coordinate non appartenenti alla spirale principale, è possibile può spostarsi sulla superficie in maniera elicoidale aggiungendo la stessa quantità Δk ad entrambe le coordinate come:

$$(\tau_1, v_1) = (\tau_0 + \Delta k, mant(v_0 + \Delta k))$$

[3]La funzione mantissa o parte frazionaria, applicata a un numero reale x, restituisce il numero stesso a cui è sottratta la sua parte intera: $mant(x) = x - \lfloor x \rfloor$

Capitolo 3

Realizzazione

3.1 Software

3.1.1 Linguaggio di programmazione

Per lo sviluppo di un prototipo dello strumento presentato in questa tesi, si è utilizzato il linguaggio di programmazione *Rust*.

Rust è stato originariamente sviluppato presso Mozilla [17]. È un linguaggio moderno, ha una sintassi abbastanza agile e leggibile rispetto a concorrenti quali il C++, pur mantenendo buone performance essendo anch'esso compilato. Esso è fortemente tipizzato, orientato al corretto uso della memoria ed è in grado di rilevare molti errori e sviste nella sua gestione già in fase di compilazione. L'ecosistema di librerie è in crescita, seppur necessiti ancora di sviluppo in alcuni settori.

Le librerie esistenti sono comunque sviluppate a sufficienza per questo progetto. Con questa scelta è stato possibile mantenere buone prestazioni, utili per un futuro utilizzo interattivo dell'algoritmo, e al contempo una buona resistenza a errori umani, avendo un sistema di controllo dei tipi molto stringente già in fase di compilazione, assente nel caso fosse stato scelto un linguaggio completamente interpretato.

La scelta è ricaduta su questo linguaggio anche per la sua buona integrazione[1] con *Godot Engine*, un motore per videogiochi. Ciò pone le basi per un futuro strumento con interfaccia grafica per la modellazione e la visualizzazione di istruzioni visive per fornire un'anteprima e assistere l'artigiano.

Librerie utilizzate

Le librerie dell'ecosistema Rust possono essere scaricate tramite un *package manager*.

Il package manager di Rust, *cargo*, permette di scaricare le dipendenze del progetto da un ampio catalogo di librerie, in maniera simile a *npm* per Javascript e *pip* per Python.

[1] Godot Engine consente l'integrazione tramite GDNative, funzionalità che permette di caricare librerie condivise compilate separatamente in altri linguaggi senza necessità di ricompilare l'intero engine [18].

Nome libreria	Descrizione	Licenza
nalgebra	Algebra lineare, operazioni con trasformazioni, matrici e vettori	Apache 2.0
petgraph	Costruzione e navigazione di grafi	Apache 2.0
lazy_static	Macro per ottenere variabili statiche il cui valore è assegnato a runtime alla prima inizializzazione, per poi essere mantenuto alle letture successive	MIT (expat)
godot-rust	Integrazione linguaggio Rust in Godot Engine	MIT (expat)

Tabella 3.1: Dipendenze software

Sarà sufficiente specificare le librerie necessarie e la versione richiesta all'interno del file `Cargo.toml`. Nel compilare il progetto, verranno risolte automaticamente le dipendenze, scaricandole dal *package registry* ufficiale.

Le librerie utilizzate sono riportate in Tabella 3.1.

Librerie scartate

L'idea iniziale di utilizzare la libreria *compas* per Python e Blender per la modellazione, tramite un plugin, è stata scartata. La libreria in esame è sicuramemente dotata di molte funzionalità e ha una integrazione con Blender; tuttavia, è troppo di alto livello e la documentazione non è molto esaustiva. Inoltre, per funzionare con Blender è richiesta una lunga procedura di setup da seguire per sostituire l'interprete Python integrato in Blender con uno di *conda*, complicando la riproducibilità di questa ricerca da parte di altri ricercatori e ostascolando l'integrazione della libreria sviluppata con altri software. In ultimo, Blender è un software avanzato con una interfaccia non adatta ai neofiti e, dunque, non calzante all'obiettivo futuro di creazione di uno strumento con interfaccia grafica utilizzabile accessibile.

Per gli stessi motivi di riproducibilità dello studio e di mantenimento del requisito di accessibilità di un futuro software a utenti meno esperti sono state scartate altre alternative quali CAD molto settoriali oppure altri con licenza proprietaria quali Rhino/Grasshopper.

3.1.2 Organizzazione del progetto

Il linguaggio Rust supporta un comportamento OOP[2] tramite una combinazione di `struct` e blocchi `impl`. Le struct, simili a quelle presenti in C, sono assimilabili a delle raccolte di variabili relative all'oggetto. A ogni `struct` possono essere associati più metodi tramite la *keyword* `impl`. Sulla stessa `struct`, a sua volta, è definibile più di un blocco `impl`. I metodi ricevono `&self` come primo argomento, ossia un riferimento all'oggetto su cui il metodo stesso è stato chiamato[3].

[2]Object Oriented Programming, paradigma di programmazione orientata agli oggetti.

[3]Il riferimento viene passato tramite il sistema di *borrowing* di Rust, e restituita l'*ownership* al chiamante al termine dell'esecuzione del metodo.

```
struct Test
{
    //... campi
}
impl Test
{
    // Metodi
}
```

Trait

I `trait` sono una raccolta di metodi non implementati, sulla falsariga delle interfacce nel paradigma OOP. Definito un `trait` "NomeTrait" e la classe "Test" dell'esempio precedente, si può implementarlo specificando i metodi in un blocco `impl` aggiuntivo:

```
//Definizione trait, adoperabile anche in altri file
trait NomeTrait
{
    fn test_function(&self) -> f64;
}

// Implementazione del trait
impl NomeTrait for Test
{
    // Implementazione dei metodi
    fn test_function(&self) -> f64
    {
        //Effettiva implementazione
        todo!();
    }
}
```

Diagramma UML

Si riporta in appendice il diagramma UML del progetto, utile a chi tentasse di espandere ulteriormente questo ambito di ricerca. Il progetto è organizzato in alcuni moduli dedicati a fasi distinte del processo, dalla modellazione alla visualizzazione. Ogni modulo espone alcune interfacce: se si desiderasse aggiungere un nuovo algoritmo di calcolo o primitive di modellazione parametrica, sarà sufficiente implementare l'interfaccia corrispondente in una nuova classe.

Il modulo `geometry` riunisce tutte le classi e interfacce relative alle primitive di base per creare la curva di controllo e la curva da estrudere. La curva di controllo allo scopo di questa tesi è una Bézier, la quale implementa l'interfaccia `Curve`. Questa fornisce alcuni metodi per il calcolo del sostegno della curva, della rappresentazione in parametrizzazione intrinseca $f(t(s))$ e restituisce i versori del sistema di Frenet-Serret e quello indipendente dalla torsione precedentemente descritto. L'interfaccia `Shape`, invece, definisce le curve $Q(s)$ trasversali che saranno utilizzate per delineare la superficie.

Il modulo `param_surface` definisce l'interfaccia `UVSurface` e le classi da essa derivate dedicate alla gestione di una superficie parametrica; inoltre, definisce il tipo di dato `TauCoords`, utile per gli spostamenti in direzione elicoidale. L'interfaccia appena citata prevede metodi per sfruttare la riparametrizzazione `TauCoords` per ottenere informazioni sulla superficie (ad esempio, restituendo il versore normale alla superficie date le coordinate parametriche), altri per il calcolo dei vertici utili alla conversione in mesh e alcune informazioni di debug. Ricordando il requisito della lavorazione a spirale, contiene anche metodi per permettere lo spostamento lungo la direzione della spirale di una distanza nota. Infine espone metodi per il calcolo della lunghezza di una geodetica[4]. Ad esempio, il metodo `width_increment(coord, dist)` restituisce le coordinate ottenute muovendosi sulla superficie da `coord` di una distanza `dist` nella direzione locale della spirale. Utilizzando la derivata, è utile principalmente per piccoli spostamenti.

$$(\tau_1, v_1) = (\tau_0 + dist \cdot \frac{d\tau}{ds}(\tau_0, v_0), v_0 + dist \cdot \frac{d\tau}{ds}(\tau_0, v_0))$$

Infine, la coordinata v viene nuovamente portata tra 0 e 1 tramite l'operatore mantissa. Similmente, esiste anche un metodo analogo per lo spostamento lungo la direzione della curva di controllo.

Nel modulo `nodes` sono definiti gli algoritmi che effettuano la generazione e il posizionamento delle maglie operanti su una superficie parametrica. Viene definita l'interfaccia `KnittableGraph`, la quale fornisce metodi per la analisi del grafo generato consentendo la sua successiva analisi a scopo di visualizzazione e generazione delle istruzioni, rispettivamente implementati nei moduli `meshviz` e `instructions`.

Tipo `Option<T>`

Al contrario di molti linguaggi, Rust non consente di restituire valori vuoti se il compilatore si aspetta un tipo specifico. Per esempio, se una funzione deve restituire il tipo `f64`, un tipo diverso, anche un valore vuoto, è rifiutato dal compilatore. Se si pianifica che una funzione possa restituire valori vuoti, è necessario indicare `Option<T>` come tipo restituito nella sua definizione. Il tipo `Option<T>` incapsula questo comportamento obbligando il programmatore a scegliere esplicitamente come trattare entrambi i casi.

I valori consentiti sono:

- `Some(e)`: un valore `e` di tipo `T` è presente e può essere utilizzato;
- `None`: il valore è vuoto. Il programmatore deve indicare come agire, fornendo un valore di default, terminando il programma o un comportamento personalizzato.

Tipo `Result<T, E>`

Il tipo `Result<T, E>` è utile per definire azioni che, seppur non restituiscano valori vuoti (dunque non adatte al tipo Option), potrebbero però fallire.

I due tipi generici da fornire sono:

[4]La geodetica è la curva più breve che congiunge due punti su uno spazio, in questo caso una superficie.

- `T`: il tipo che si intende ottenere in caso di esito positivo
- `E`: un tipo che conservi l'informazione sull'errore avvenuto, la sua tipologia o alcuni dati utili per il debug. È possibile definire anche un tipo errore personalizzato a patto di implementarne i metodi principali.

In questa tesi è stato definita una classe di errori `CrochetError` come indicato nel diagramma in appendice.

Test

Il software sviluppato adopera le funzioni di test del linguaggio Rust. Sono presenti *doctest* in linea per verificare la correttezza di singole funzioni e *integration test*, che combinano i moduli in verifiche a tutto tondo.

I doctest sono definiti tramite tripla barra prima della definizione di una funzione o di un metodo. La tripla barra serve per indicare al compilatore che il testo non è un commento per lo sviluppatore, ma deve essere incluso nella pagina di documentazione. Esso deve essere scritto in formato markdown. Quando è inserito del codice in markdown, esso viene compilato come un programma a sé stante ed eseguito in fase di test. Sono presenti doctest per funzioni interne per assicurare la correttezza del calcolo delle metriche e del funzionamento delle primitive geometriche, curve e superfici.

Gli integration test, invece, sono definiti esternamente ai metodi: in questa tesi, ognuno di essi dichiara una superficie con caratteristiche diverse, alcune semplici tronchi di cono, altre più elaborate con curvature più estreme, lanciati con entrambi gli algoritmi e variando inoltre la dimensione delle maglie.

3.2 Algoritmo proposto

In questa sezione si presenta l'algoritmo ideato per generare il grafo che sarà convertito in istruzioni di cucitura successivamente.

L'algoritmo opera in alcune fasi, elencate di seguito:

1. Inizializzazione:
   - definizione delle dimensioni target delle maglie;
   - definizione della superficie target in modo parametrico.
2. Posizionamento e collegamenti nodi:
   - prima analisi e inizializzazione del primo giro del grafo;
   - calcolo distanza e numero di maglie necessario;
   - ottimizzazione numero coordinate per garantire la cucibilità delle short row;
   - valutazione della necessità di aumenti o diminuzioni;
   - effettiva aggiunta al grafo dei nodi necessari;
   - collegamento della *patch* della colonna generata ai nodi precedenti.
3. Produzione formato stitch mesh per la visualizzazione;
4. Produzione istruzioni.

3.2.1 Inizializzazione

Definizione dimensione delle maglie

La scelta delle dimensioni delle maglie dipende dal tipo e dallo spessore del filo utilizzato, nonché dalle caratteristiche stilistiche dell'artigiano nel regolare la tensione del filo. È consentito definire un valore di larghezza e altezza diversi tra loro e indipendenti. Tramite questa scelta, si tiene conto delle specificità dello stile di ogni artigiano in quanto si permette la rappresentazione di maglie anche non quadrate.

Definizione superficie parametrica

La superficie parametrica è definita tramite una curva di controllo e delle curve che controllano il raggio della superficie punto per punto, definite dall'utente. Nell'implementazione attuale, la curva di controllo è una Bézier singola, ma potrà essere estesa al supporto di altri tipi di curve (MultiBézier, Spline e altre curve parametriche). Per quanto concerne le curve che definiscono il raggio della superficie, il software supporta le seguenti tipologie:

- circonferenza, dato il raggio;
- anello magico[5];
- ellisse;
- curva tramite segmenti concatenati, anche non planari.

Sono stati implementate e supportate superfici di due tipi, entrambi derivati dall'interfaccia `UVSurface`: la prima è un tronco di cono curvabile, la seconda permette di esprimere superfici UV generiche. Il tronco di cono viene definito dalla curva di controllo, dal raggio della superficie all'inizio e alla fine di essa. Quella più libera, invece, è realizzata tramite una classe denominata `MultiShapeSurface` che consente di adoperare curve arbitrarie per la definizione del raggio della superficie in ogni punto. A livello implementativo, viene inizializzata da un elenco di curve e i valori di u a cui porle sulla curva di controllo. Il suo metodo costruttore accetta qualsiasi tipo di curva per definire il raggio, a patto che implementi l'interfaccia comune `Shape`.

3.2.2 Struttura dati

L'algoritmo esposto produce un grafo diretto successivamente visitabile a scopo di visualizzazione o generazione istruzioni.

Vertici ed archi mantengono un peso, ossia una struttura dati associata, con alcune informazioni. La struttura dati associata ai vertici mantiene in memoria:

- `TauCoords`, coordinate della posizione del punto nel sistema di riferimento parametrico;

[5] L'anello magico, o *magic ring*, è una circonferenza tale che, cucita, sarà composta da esattamente 6 maglie. Il raggio è calcolato durante l'esecuzione del programma in base alle dimensioni della maglia impostata precedentemente.

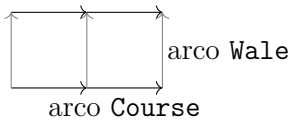


Figura 3.1: Un grafo in assenza di short row

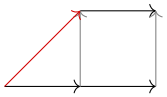


Figura 3.2: Un grafo con inizio short row: in rosso l'arco `CourseNextOverrideStart` a cui viene data priorità

- `short_row_level`, il quale vale 0 se si tratta di un nodo base appartenente alla spirale principale, altrimenti un numero naturale indicante il livello di short row in analisi;
- `target_tau`, l'altezza target da raggiungere alla prossima analisi del vertice.

Gli archi, invece, mantengono un tipo enumerativo con i seguenti valori:

- `Wale`
- `Course`
- `CourseNextOverrideStart`
- `CourseNextOverrideEnd`

Gli archi `Wale` collegano vertici nella direzione di ordito, ossia in senso longitudinale alla curva di controllo. Gli altri archi rappresentano collegamenti in direzione trasversale alla curva di controllo. Il percorso ottenuto seguendo gli archi `Course` in sequenza approssima l'elica costruita intorno alla superficie. Un collegamento standard è mostrato in Figura 3.1.

Le varianti `CourseNextOverrideStart` e `CourseNextOverrideEnd` sono necessarie quando vengono aggiunte le short row. In particolare, `CourseNextOverrideStart` tiene traccia del suo inizio e `CourseNextOverrideStart` della sua fine. Durante lo scorrimento in direzione della trama (`Course`), se l'algoritmo trova sia un arco di tipo `Course`, sia un arco con `Override`, seguirà il secondo. Per esempio, nella Figura 3.2 verrà data priorità e sarà seguito l'override in rosso.

Anche `CourseNextOverrideEnd` ha la stessa priorità di `CourseNextOverrideStart`.

3.3 Posizionamento e collegamento dei nodi

In questa fase, viene analizzata la superficie definita al passo precedente e viene realizzato un grafo che rappresenti i collegamenti del tessuto calcolati. L'algoritmo presentato e quello di confronto sono implementati come due classi distinte, rispettivamente denominate `CrochetGraph` e `CapunamanPaperGraph`. Essi differiscono nel metodo di calcolo del grafo,

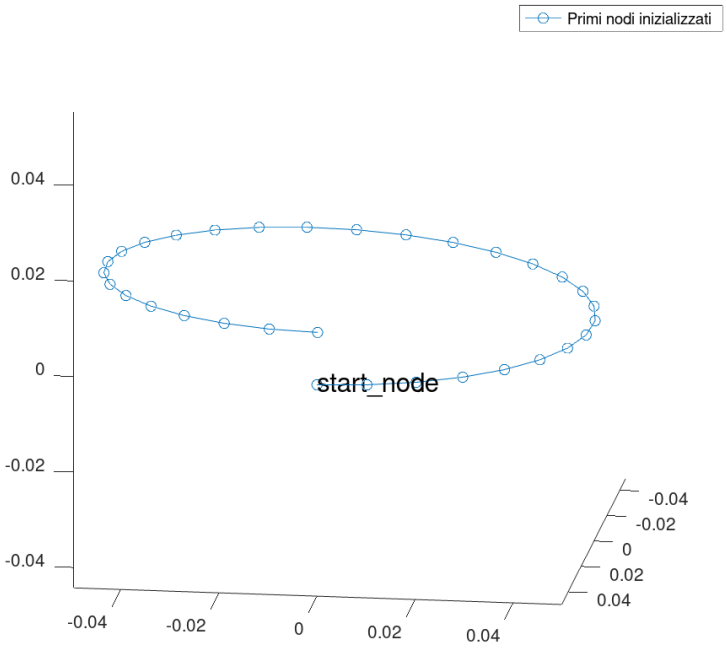


Figura 3.3: Inizializzazione del grafo

ma condividono numerosi metodi utili al loro controllo e impiegati nel calcolo delle metriche atte a confrontarli. Perciò, le due classi implementano una interfaccia comune, definita tramite il `trait` `KnittableGraph`.

Prima analisi

Quando un oggetto `CrochetGraph` viene istanziato tramite il suo costruttore, ricevendo la superficie parametrica come argomento, esso dapprima stima la lunghezza del sostegno dell'elica sulla superficie S di $E(\tau) := S(\tau, mant(\tau))$ per τ tra 0 e 1, ossia $\int_0^1 S'(r, r)dr$. La distanza calcolata viene divisa per la larghezza della maglia impostata, in modo da ottenere il numero di nodi di partenza. Questi nodi vengono già salvati all'interno del grafo e collegati con degli archi di tipo `Course`.

In Figura 3.3 si mostra il grafico appena inizializzato: il primo nodo che è stato aggiunto viene tenuto in memoria come `start_node`, utile per il passo successivo dell'algoritmo. Ogni nodo generato è legato al successivo con un arco diretto di tipo `Course`.

3.3.1 Calcolo delle coordinate dei nuovi punti

L'algoritmo opera localmente tramite una finestra scorrevole di larghezza personalizzabile. A ogni iterazione, la finestra viene fatta scorrere di un nodo in avanti in direzione `Course`. Qualora ci fosse sia un arco `Course`, sia un arco Override, la priorità verrà data al secondo,

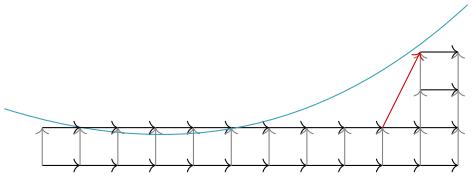


Figura 3.4: Colonne di nodi generate tramite singola soglia

come esposto precedentemente. Per ogni nodo di base nella finestra, verrà generata una colonna di nodi su di esso, con numero di elementi variabile, in direzione `Wale`.

La struttura dati della finestra è così definita:

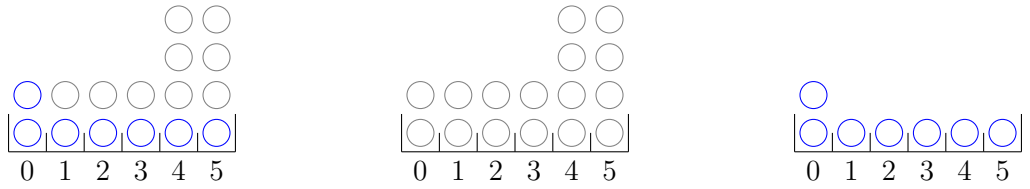
```
struct BufferCreateNodes
{
    columns: [Vec<TauCoords>; 6],
    nodes: [Vec<NodeIndex>; 6],
}
```

Essa mantiene separate le informazioni sulle coordinate alle quali dovranno essere generati i nuovi nodi dagli indici dei nodi già generati e aggiunti al grafo. Il primo array `buffer.columns` contiene le coordinate precalcolate dei vertici che dovranno essere aggiunti al grafo; dopo che il loro numero e posizionamento è stato ottimizzato, essi sono aggiunti al grafo. Nel secondo array `buffer.nodes` sono contenuti gli indici dei vertici confermati ed effettivamente generati. Ciò è computazionalmente vantaggioso, in quanto vengono evitate operazioni superflue di aggiunta e rimozione nodi dal grafo. Riguardo l'aspetto algoritmico, poiché i vertici sono salvati in sequenza, la rimozione di un nodo dal grafo causa uno slittamento degli indici, invalidandoli; l'applicazione anticipata su `buffer.columns` delle eliminazioni necessarie durante l'ottimizzazione limita il numero di quelle eseguite sul grafo. Si riporta in Figura 3.5 una rappresentazione dello stato dei due array.

Logica a singola soglia

Nella suo principio base di funzionamento, l'algoritmo stima la distanza tra il nodo in analisi nella finestra e la sua altezza target (in azzurro nella Figura 3.4). Utilizzando la altezza dell'arco `Wale` ideale impostata in fase di inizializzazione, l'algoritmo verifica quanti nodi dovrano essere generati in direzione `Wale`. Il numero di vertici in ogni colonna deve essere pari, allo scopo di mantenere il requisito di cucibilità delle short row, dettagliato successivamente; non esistono colonne con un numero dispari di vertici.

Al contempo, è necessario mantenere un numero di nodi maggiore di uno in ogni colonna. Infatti, la cucitura avviene come una unica spirale non interrompibile. Di conseguenza, qualora l'algoritmo si trovasse nella situazione sulla sinistra della figura sopracitata in cui la distanza da coprire, rappresentata dalla soglia in azzurro, è minore della altezza della maglia impostata, la generazione del nodo verrebbe forzata allo scopo di garantire la validità della struttura.



(a) (b) (c)

Figura 3.5: Esempio di stato degli array `buffer.columns`, in grigio, e `buffer.nodes`, in blu; (a) stato di entrambi gli array, sovrapposti; (b) i soli dati dell'array `buffer.columns`; (c) i soli dati dell'array `buffer.nodes`.

Logica a doppia soglia

L'uso di una singola soglia come illustrato finora introdurrebbe numerose short row di brevissima lunghezza. Queste vengono filtrate e rimosse, ma l'algoritmo qui illustrato porta un ulteriore miglioramento.

Analizzando la letteratura disponibile, l'algoritmo incrementale recente di Narayanan [11] argina il problema scartando le short row troppo brevi, ossia con una lunghezza minore di una lunghezza specificata, in quel caso arbitrariamente fissata a quattro maglie. Ciò semplifica il processo di cucitura e migliora l'aspetto estetico del risultato, poiché le facce di inizio e fine short row sono visibili e devono essere adoperate con parsimonia. Tuttavia, la logica usata non risulta essere sempre la soluzione ottimale.

Si analizzano le situazioni possibili, come le due logiche a singola soglia si comportano e si propone un algoritmo migliorativo.

Le due situazioni in cui si notano le differenze algoritmiche sono delineate nelle due colonne nelle immagini nella Figura 3.6. La colonna di sinistra riassume il caso in cui la distanza dei nodi rispetto alla altezza target sia approssimabile a quella raggiunta da una short row intera. La colonna di destra, invece, mostra il caso in cui i due picchi del valore della funzione siano isolati, quindi punti trascurabili.

Il primo algoritmo, usato da Narayanan [11], elimina le short row troppo brevi. Ciò fornisce un risultato ideale nella situazione riportata in Figura 3.6b, ma non in quella in Figura 3.6a. L'uso di una singola soglia rigida introduce del rumore che fa rimuovere l'intera short row, nonostante la ridotta variazione della funzione.

Per arginare il problema in maniera simile, si potrebbe ipotizzare di forzare la generazione di una short row unica quando ne sono presenti due brevi molto ravvicinate. Questo metodo però, al contrario, mostra un comportamento ottimale nella situazione mostrata in Figura 3.6c, ma non nel caso in Figura 3.6d, poiché non è in grado di distinguere l'entità della funzione distanza (in blu) tra i due picchi ed è rigidamente vincolato al superamento della singola soglia impostata.

In ultimo, l'approccio presentato in questa tesi prevede una analisi preventiva dei nodi del giro corrente guidato da due soglie. L'esito è un comportamento corretto in entrambe le situazioni.

L'approccio presentato in questa tesi utilizza due soglie, denominate `SOGLIA_ATTIVAZIONE` e `SOGLIA_TOLLERANZA`. La prima è inizializzata al valore 3, la seconda al valore 2, entrambe regolabili a piacere. La seconda regola la sensibilità dell'algoritmo: quando

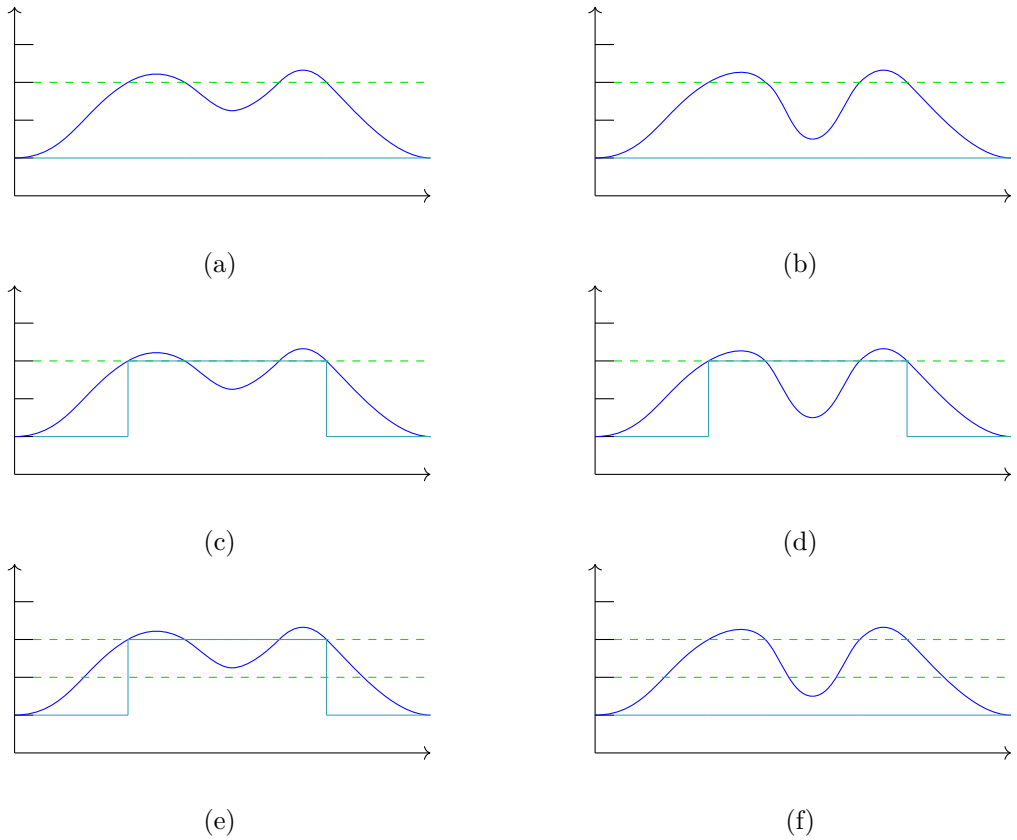


Figura 3.6: Confronto approcci di ottimizzazione delle short row, in due situazioni tipo; sull'asse delle ascisse avanzano i nodi della spirale, sull'asse delle ordinate la linea blu mostra il numero reale di maglie necessarie per raggiungere il livello successivo della spirale target, in verde è mostrata la soglia utilizzata, in azzurro le zone che l'algoritmo decide di conservare.

tende a un valore vicino a 1, vengono privilegiate *short row* uniche lunghe, a costo di sforare molto l'altezza target. Al contrario, un valore vicino a `SOGLIA_ATTIVAZIONE` porterà a un comportamento più simile agli algoritmi a soglia rigida precedentemente illustrati.

Viene dapprima calcolato il valore di maglie idealmente da generare sopra al nodo attuale, come un numero reale.

```
let maglie: f64 = (coord_up.tau - coord_down.tau).signum()
                  *
                  (surf.geodesic_dist(coord_down, coord_up, None).unwrap()
                  /
                  crochet_consts::get_height());
```

Il numero delle maglie (secondo fattore della moltiplicazione) è moltiplicato per il risultato di `signum()`, che restituisce 1, 0, o -1. Se non fosse presente questa accortezza, il metodo genererebbe *short row* irragionevoli quando il nodo ha già superato di molto il valore del proprio target.

Se il numero risultante è maggiore della `SOGLIA_ATTIVAZIONE`, inizia un procedimento di *look ahead* con cui l'algoritmo, ancor prima di generare i nodi, pre-analizza i passi successivi con la condizione illustrata nelle Figure 3.6e e 3.6f e costruisce un array che guiderà il numero di short row.

La short row viene mantenuta attiva finché il valore di maglie calcolato è maggiore della seconda soglia, ossia `SOGLIA_TOLLERANZA`.

Nel momento in cui il valore di maglie calcolato diventasse minore anche della seconda soglia, verrebbero aggiunti all'array da restituire al chiamante tutti i nodi precedenti analizzati fino all'ultimo per cui il numero di maglie sulla propria colonna sia maggiore di `SOGLIA_ATTIVAZIONE`. Gli ultimi nodi per cui `maglie > SOGLIA_TOLLERANZA` ma `maglie < SOGLIA_ATTIVAZIONE` vengono scartati.

Gestione aumenti e diminuzioni

Con il procedimento mostrato finora, non è possibile aumentare o diminuire la curvatura della superficie in direzione trasversale alla curva di controllo. Per ottenere questo risultato, vengono aggiunti aumenti e diminuzioni. In fase di analisi del vettore `buffer.columns`, viene calcolata la distanza tra i vertici allo stesso livello. Per esempio, si confronta la distanza tra `buffer.columns[0][1]` e `buffer.columns[1][1]`. Se essa supera una soglia s_0 definita in funzione della larghezza della maglia desiderata, viene aggiunto un vertice supplementare. Viceversa, se la distanza tra i due vertici è minore di una seconda soglia s_1 (quando essi sono troppo vicini), vengono collassati in un unico punto, dando luogo a una diminuzione. Per l'aumento, il calcolo delle coordinate dei due punti avviene tramite interpolazione lineare sulla superficie delle `TauCoords`, il primo con fattore 0.33, il secondo con fattore 0.66. Infine, per quanto concerne le diminuzioni, viene restituito un vettore con il solo elemento di base. Tramite questa scelta, la fase successiva non genererà nuovi nodi nel grafo, bensì si limiterà a collegare quelli esistenti. È d'uopo impedire la generazione di due diminuzioni consecutive e non intervallate da una maglia singola, pena la non validità del grafo risultante.

Per migliorare la fedeltà alla superficie, distribuendo gli aumenti e le diminuzioni prioritariamente nelle zone in cui la curvatura cambia, e mantenendo la maglia semplice nei punti piani, si integra nel confronto il seguente fattore aggiuntivo. Dati i punti di un quadrilatero definiti in senso orario A, B, C, D, si calcola la differenza percentuale di lunghezza tra $\vec{b_1} = D - A$ e $\vec{b_2} = B - C$.

$$r = \frac{K \cdot (|\vec{b_2}| - |\vec{b_1}|)}{|\vec{b_1}|}$$

Il valore di r si avvicina a 0 quando la superficie non si sta né allargando, né stringendo; è maggiore di 0 durante un allargamento della superficie e minore di 0 durante un suo restringimento. La costante K determina una scalatura del valore e influenza la sensibilità dell'algoritmo alla variazione di curvatura. Impostarla a $K = 0$ disattiva il comportamento descritto. Il valore di r è usato direttamente nella soglia. Per esempio, viene determinato un aumento quando:

$$dist + dist \cdot r > s_0$$

Inoltre, poiché l'operazione di disposizione di punti per il cucito rappresenta un sottocampionamento spaziale della superficie, è utile inserire un piccolo segnale di *dithering* per

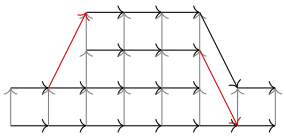


Figura 3.7: Short row rappresentate nella struttura a grafo

migliorare l'accuratezza del risultato. Alla distanza *dist* utilizzata nel confronto, è possibile aggiungere un segnale S_d definito come segue:

$$S_d = A \cdot sin(2\pi f v)$$

dove:

- S_d è il segnale risultante;
- A è una costante moltiplicativa, utile per modificare l'ampiezza del segnale;
- f è un moltiplicatore per variare la frequenza;
- v è la seconda coordinata del punto in esame sulla superficie parametrica data.

Nell'algoritmo proposto, si richiede che $f \in \mathbb{N}$ per impedire discontinuità di S_d in corrispondenza di $v = 0$ e $v = 1$.

Short row nel grafo

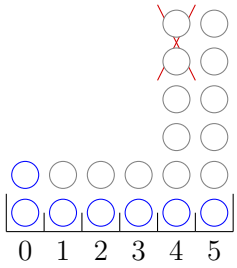
La Figura 3.7 mostra come una short row sia conservata all'interno del grafo, con quali collegamenti e ruolo dei nodi. Gli archi di tipo `CourseNextOverrideStart` e `CourseNextOverrideEnd` sono denotati in rosso, rispettivamente il primo sulla sinistra e il secondo sulla destra.

Si rammenta che, per mantenere la cucibilità, non è possibile inserire differenze nel numero di nodi superiori a due tra due colonne adiacenti. Di conseguenza, nel caso mostrato in Figura 3.8a, il salto viene livellato portando a quattro il numero di nodi nella colonna intermedia di passaggio.

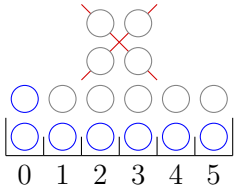
L'altra operazione applicata è la rimozione di short row troppo brevi. La decisione di effettuare questa operazione è stata presa per limitare il numero delle facce di inizio-fine short row, in quanto complicano il lavoro all'artigiano e sono esteticamente differenti rispetto alla maglia standard. Sono fondamentali per condizionare l'andamento della superficie risultante, ma sono da usarsi con parsimonia e perizia nel loro posizionamento per non costituire uno svantaggio nella resa finale.

Connessione patch

Dopo che sono stati generati e salvati all'interno del grafo i vertici della colonna in analisi, essi devono essere collegati alla struttura generata nei passi precedenti, come nelle immagini in Figura 3.9. In magenta sono indicati gli archi aggiunti in questa fase.

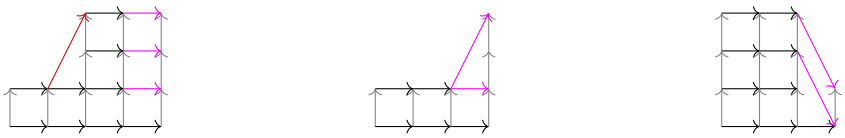


(a) Un salto di più di due nodi viene impedito



(b) Short row troppo breve

Figura 3.8: Fase di ottimizzazione delle short row



(a) (b) (c)

Figura 3.9: Connessione patch colonna generata ai nodi precedenti; (a) caso in cui le colonne abbiano pari numero di nodi; (b) caso di inizio short row; (c) caso di fine short row.

Vengono utilizzati semplici archi di tipo `Course` qualora la colonna generata e quella precedente abbiano lo stesso numero di vertici, come in Figura 3.9a.

In alternativa, quando una delle due colonne ha un numero di elementi maggiore di 2 rispetto all'altra, vengono aggiunti anche archi di tipo Override per marcare l'inizio o fine delle short row (Figure 3.9b e 3.9c).

3.4 Algoritmo di riferimento

Terminata l'illustrazione del metodo di calcolo del grafo dell'algoritmo proposto, si descrive l'approccio di quello di riferimento, sempre applicato a una superficie parametrica, di cui si possono distinguere le fasi elencate di seguito:

1. Definizione superficie parametrica e dimensione maglia desiderata (W,H);
2. Campionamento con isocurve in direzione U;
3. Formazione di una spirale;
4. Campionamento in direzione V;
5. Collegamento nodi.

La prima fase è comune a entrambi gli algoritmi e l'averla mantenuta consistente assicura la confrontabilità dei risultati, poiché derivati dalla stessa superficie, definita allo stesso modo.

Il lavoro di riferimento dapprima campiona la superficie in direzione U, ottenendo delle isocurve, purtroppo senza però illustrare i dettagli dell'operazione. Data l'altezza della maglia desiderata H, possono essere applicate diverse tecniche per il campionamento. Per esempio, H può essere adoperata come distanza minima tra le isocurve, massima, o media. Allo stesso modo, non è specificato se H sia confrontata con la distanza proiettata sulla curva di controllo o con quella geodetica[6], o altro. Le tecniche elencate collassano allo stesso risultato nel caso di un semplice cilindro, ma giungono a risultati diversi in linea generale. Di conseguenza, è stata mantenuta la stessa strategia usata nell'algoritmo ideato in questa tesi, ossia di imporre H pari alla distanza minima tra ogni coppia di isocurve.

Poi, congiungendo gli estremi di ogni isocurva al livello successivo, viene formata una spirale unica. Ogni giro della spirale lungo l è campionato sul parametro V tramite la dimensione impostata W, ottenendo $\frac{l}{W}$ nodi.

La fase di collegamento nodi itera su ogni nodo generato e, per ogni giro, ne confronta la posizione con quelli presenti al giro successivo. Quando viene trovata la coppia di nodi con distanza minore, essi vengono collegati con un arco di tipo `Wale`. Per questa valutazione, viene utilizzata la lunghezza della geodetica.

In seguito, è necessario integrare gli archi di tipo `Course` per rendere il grafo percorribile e con una struttura valida. L'obiettivo viene raggiunto iterando sull'elenco dei nodi, nell'ordine in cui sono stati generati, e collegando ognuno di essi al successivo.

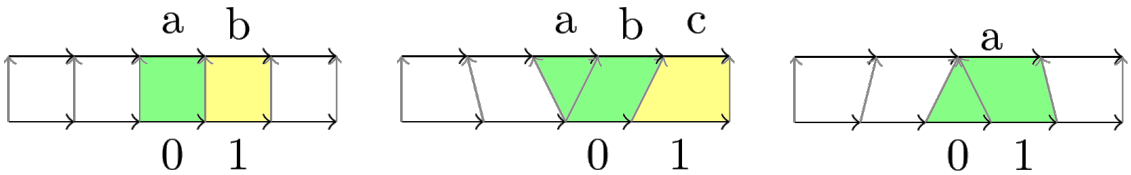
3.5 Generazione istruzioni testuali dal grafo

L'output della prima parte dell'algoritmo è un grafo: esso viene percorso nuovamente dal generatore istruzioni testuali. I due algoritmi che si intende confrontare sono implementati come due classi separate. Tuttavia, i loro risultati possono essere impiegati nelle fasi successive di generazione istruzioni e visualizzazione. Lo strumento software è stato progettato in modo che le due classi in esame implementino entrambe l'interfaccia `KnittableGraph`, offrendo dei metodi comuni standardizzati. In seguito, la fase di generazione istruzioni si aspetta in input un oggetto di una qualsiasi classe che implementi quell'interfaccia come `trait`. Il comportamento è ottenuto impostando il tipo dell'argomento richiesto a `Box<dyn KnittableGraph>`[7].

Le istruzioni generate sono espresse tramite il tipo enumerativo `StitchType`. Le relazioni tra di esse, invece, tramite `StitchConnection`, presente nel modulo `meshviz`.

[6] Una geodetica è la curva di lunghezza minima che unisce due punti in uno spazio metrico. Lo spazio può essere, in questo caso particolare, una superficie.

[7] L'operatore `Box` è un puntatore a un valore allocato nella memoria *heap* che, quando esce dallo *scope* in cui era stata dichiarata, procede automaticamente alla distruzione dell'oggetto a cui punta. La parola chiave *dyn* indica al compilatore che, poiché esso si aspetta un `trait` implementabile da classi diverse, la dimensione della struttura effettivamente ricevuta a runtime non è nota in fase di compilazione.



(a) (b) (c)

Figura 3.10: Area delle maglie generate dal grafo; (a) maglia bassa; (b) aumento; (c) diminuzione.

Il modulo consta di una struttura a finestra che viene fatta scorrere arco per arco, seguendo l'elica principale e costruendo un array di oggetti istruzione. Questi sono successivamente trasformati nella abbreviazione testuale corrispondente e accorpati per migliorarne la leggibilità.

Come dato aggiuntivo, da ogni maglia viene desunto un arco come "direzione Wale primaria", utilizzata per lo scorporamento dei giri dettagliato nella Sezione 3.5.3.

3.5.1 Logica di base

Incominciando dal caso più semplice in cui non siano presenti short row, l'algoritmo viene inizializzato con una finestra di dimensione 2, in cui vengono inseriti i primi due archi di tipo `Course` del grafo. Essa, a ogni passaggio, verrà fatta proseguire, trovando il nuovo arco `Course` e scartando quello all'indice 0, ormai non più utile. L'algoritmo termina quando non sono più presenti archi di quel tipo e si trova dunque al termine del grafo.

Come risultato temporaneo, viene trovato l'elenco ordinato degli archi `Course` che connettono i nodi collegati in direzione `Wale` a quelli di base. La Figura 3.10 illustra le condizioni per generare le tre tipologie principali di maglie. Si illustra di seguito la logica decisionale.

Maglia bassa

Questa è la maglia più semplice, in cui ogni nodo ha un singolo arco `Wale`, e i nodi destinazione dei due archi non coincidono. La Figura 3.10a delinea la situazione. All'arco 0 è assegnato l'arco "a", all'arco 1 è assegnato l'arco "b". La maglia generata è evidenziata in verde, viene generata una maglia bassa, la cui rappresentazione testuale è *mb* nella notazione italiana o *sc*, abbreviazione di *single crochet* in inglese. Le informazioni dell'arco 1 sono ignorate poiché saranno utilizzate alla successiva iterazione dell'algoritmo. La finestra scorre di una unità.

Aumento

Un aumento incrementa il numero di nodi tra due giri di cucito. Sono presenti nodi collegati a due nodi successivi in direzione `Wale`.

Lo stato viene descritto in Figura 3.10b; all'arco 0 sono assegnati in ordine "a" e "b", mentre all'arco 1 viene assegnato l'arco "c". L'area evidenziata in verde è la maglia generata a cui corrisponde un aumento, la cui rappresentazione testuale è *aum* nella notazione

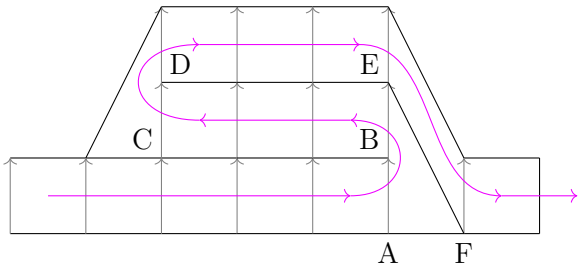


Figura 3.11: Ordine di cucitura delle maglie calcolate

italiana o *inc* in inglese. Anche qui le informazioni di 1 sono ignorate e la finestra scorre di una sola unità.

Diminuzione

In conclusione, una diminuzione consente di ridurre il raggio locale della superficie, tramite l'unione di due nodi di un livello in uno solo a quello successivo.

La Figura 3.10c delinea la situazione. All'arco 0 è assegnato un array vuoto, mentre all'arco 1 si associa l'arco "a". La maglia risultante è evidenziata in verde. È stata generata una diminuzione, la cui rappresentazione testuale è *dim* nella notazione italiana o *dec* in inglese. L'area della diminuzione adopera entrambi gli archi di base per cui, allo scopo di evitare una compenetrazione delle facce al passo successivo e un doppio conteggio dell'arco "1", la finestra scorre di due unità in questo caso.

3.5.2 Risoluzione short row

Per cucire le maglie generate, è necessario mantenere il verso della spirale e assicurarsi di non interrompere il filo. Di conseguenza, le short row sono generate a coppie di due, per mantenere la direzione di cucitura. Il concetto è illustrato in Figura 3.11 in cui al grafo è sovrapposta la direzione di risoluzione delle maglie.

Essa è rilevata tramite la verifica della presenza di un arco di fine short row nel grafo. Dopo che l'algoritmo è giunto ad analizzare fino al vertica A, vengono identificati i vertici che marcano l'inizio e la fine della short row. Sono quindi calcolate le maglie esistenti tra C e B, poi invertite in modo da seguire temporaneamente la direzione opposta a quella di cucitura, ossia B-C. Le maglie sono salvate in un vettore, che viene poi invertito. Infine, viene ripresa la direzione normale di cucitura riavviando l'algoritmo a partire da D, passando per E e proseguendo normalmente da F in poi.

I triangoli presenti in corrispondenza dell'inizio e della fine di una short row non danno luogo a una maglia effettivamente cucita, ma godono di un valore enumerativo che li rappresenti. Quando si incontra una short row e avviene un cambio di direzione temporaneo, esso deve essere esplicitato anche nelle istruzioni generate. L'informazione è descritta tramite due valori appositi in `StitchType`:

- `TurnBack`, inverte la direzione di cucitura per percorrere la prima metà della short row;

- `TurnForward`, ristabilisce la direzione standard.

La versione testuale dei due valori corrisponde ai caratteri Unicode delle due frecce direzionali: "←" e "→".

Di seguito viene riportato un esempio di istruzioni contenti una short row.

```
6 mb
2 aum
4 mb
←
  -3 mb
  -2 aum
  -2 mb
→
4 mb
2 aum
25 mb
←
  -4 mb
  -3 aum
→
4 mb
2 aum
10 mb
```

3.5.3 Miglioramento della leggibilità

L'output della fase precedente è un array di tipo `Vec<KnittableStitch>`. Limitarsi a mostrare all'utente un elenco con la rappresentazione testuale di ogni `KnittableStitch` riga per riga sarebbe lungo (migliaia di righe), confusionario (sarebbe facile "perdere il conto") e manchevole di punti di riferimento certi da cui l'artigiano possa ri-incominciare con sicurezza in caso commettesse un errore. Si illustra quindi di seguito l'approccio utilizzato per affrontare queste limitazioni.

Accorpamento

Per ridurre il numero di righe e il rischio che l'artigiano si confonda, anziché:

```
mb
mb
aum
mb
mb
mb
mb
mb
mb
mb
```

```
mb
mb
mb
mb
```

è più conveniente scrivere:

```
2 mb
1 aum
11 mb
```

Il modulo `instructions` è dedito alla gestione delle istruzioni, integrando anche questi miglioramenti. Ciò è ottenuto creando una variabile di tipo `Vec<(StitchType, u32)>`, ossia un array di tuple che combinino il tipo di maglia da cucire e il numero di volte in cui essa deve essere ripetuta. Si itera su ogni `StitchType`: se coincide con quella precedente, ci si limita ad incrementare il contatore delle ripetizioni.

Il confronto è attuabile tramite l'operatore standard `==`, poiché `StitchType` deriva una implementazione standard del metodo di confronto tra i valori dei tipi enumerativi tramite la direttiva `#[derive(PartialEq)]` in fase di dichiarazione del tipo.

Suddivisione per giri

Quando un artigiano si accorge di aver commesso un errore durante la cucitura, disfa una parte del lavoro realizzato per tornare a un punto noto. A questo scopo, le istruzioni sono separate per "giro" della spirale. Durante la cucitura, l'inizio di ogni giro è spesso marcato tramite una spilla o del filo colorato: ciò consente di disfare solo il numero minimo indispensabile di maglie per poter ripartire da una riga nota del file istruzioni.

Per supportare questo comportamento, durante la fase di calcolo delle maglie illustrata alla Sezione 3.5.1, si tiene traccia del collegamento in direzione `Wale` tra la maglia inferiore e superiore. Viene salvata in un array la colonna di maglie create in direzione `Wale` sopra alla prima maglia esistente. Durante la generazione delle istruzioni testuali, per ogni maglia da esportare nel file di output viene verificato se essa si trovi nell'array appena costruito. Nel caso la condizione sia soddisfatta, viene aggiunto un marcatore di inizio giro.

Le istruzioni vengono opportunamente indentate rispetto ai marcatori per conservarne la leggibilità.

Di seguito viene riportato un esempio di output suddiviso con i marcatori.

```
Giro 4
    18 mb
    2 dim
    39 mb

Giro 5
    4 mb
    1 dim
    27 mb
    1 dim
    11 mb
```

```
2 dim
9 mb
```

3.6 Visualizzazione anteprima

Il risultato può essere visualizzato tramite un render di anteprima verosimile oppure in maniera schematica. Il primo metodo punta a una visualizzazione esteticamente fedele al prodotto finito. Il secondo, invece, consiste nella creazione di una mesh che evidenzi i vertici del grafo e i collegamenti tra essi. Opzionalmente, si può anche mostrare la spirale target. Ciò è utile a scopo di debug, meno per l'utente finale.

Lo strumento software sviluppato mette a disposizione dei metodi per esportare l'elenco dei dati dei vertici anche in modo testuale, permettendo ulteriori elaborazioni esterne.

3.6.1 Generazione mesh con triangolazione

Innanzitutto, è necessaria una soluzione per poter generare una mesh e visualizzarla. L'algoritmo di generazione della mesh è implementato nel codice Rust e crea l'elenco dei vertici di ogni triangolo da disegnare, avendo cura di mantenere il *winding order* in maniera coerente. Non rispettarlo genererebbe delle facce con il versore normale diretto verso l'interno dell'oggetto. Ciò non è corretto quando si usi un motore di rendering dotato di *backface culling*[8]. Nel caso di Godot Engine, l'ordine corretto è in senso orario [19].

I dati vengono passati, tramite l'integrazione GdNative, da Rust all'interno di Godot Engine come valore di ritorno. La libreria `godot-rust` si occupa della conversione dai tipi statici propri di Rust ai tipi presenti in Godot e viceversa. Godot Engine consente la definizione di mesh in maniera programmatica tramite alcune funzionalità:

- classe ArrayMesh, comoda per inizializzare una mesh da un elenco di array che descrivano le varie proprietà (vertici, indici, UV, vertex color);
- classe MeshDataTool, strumento per la modifica di ArrayMesh già esistenti;
- classe SurfaceTool, utile per costruire una mesh specificando i parametri di ogni vertice, uno alla volta in sequenza;
- classe ImmediateGeometry, adoperata per generare mesh con un basso numero di punti, ma che cambino a ogni fotogramma.

Dato il numero di punti utilizzato, potenzialmente molto alto quando il numero di maglie generate aumenta, e la natura statica dell'anteprima generata, si è deciso di utilizzare il primo strumento ArrayMesh.

Un utilizzo base della ArrayMesh prevede la definizione individuale di ogni vertice, per ogni triangolo desiderato. Tuttavia, i vertici in comune vengono duplicati, una volta

[8] Il *backface culling* è una operazione della pipeline di rendering atta a migliorare le prestazioni; tramite un prodotto scalare tra la direzione di visione della camera e il versore normale alla faccia in esame, vengono scartati tutti i vertici sul lato non visibile dell'oggetto, riducendo il carico sulla scheda grafica.

per ogni triangolo di cui fanno parte. Ciò avviene nonostante siano coincidenti, di fatto aumentanto il *vertex count* totale.

Per superare questa limitazione, si fa uso delle *indexed triangle mesh*. Il codice Rust genera due vettori, uno di tipo `Vec<Point3<f64> >` per dichiarare le posizioni dei vertici, l'altro di tipo `Vec<usize>` per quanto concerne gli indici. I vertici comuni sono definiti solo una volta nel primo vettore, per poi essere indirizzati più volte nel secondo vettore.

3.6.2 Approccio a tessere

Nel panorama della visualizzazione per la tecnica dell'uncinetto esistono alcune ricerche da tenere in considerazione. Narayanan propone un approccio a tessere componibili o *tile* [20], pur senza fornire molte indicazioni su come operare il rendering. Ciò può essere integrato con il formato *augmented stitch mesh*, ossia un grafo di una mesh in cui a ogni faccia sono associate le istruzioni macchina per produrla e, a ogni coppia di facce, un arco diretto che conservi il rapporto tra esse e la direzione. L'arco è etichettato come `Course` o `Wale`, in base al suo ruolo. La struttura così descritta può essere impiegata per programmi per la modifica interattiva [21].

Basandosi sulla tecnica a tessere, l'algoritmo sviluppato genera una mesh poligonale di anteprima, esportabile in formato standard `.stl`.

La pipeline di generazione del modello 3D è strutturata nei passaggi seguenti:

- definizione tessere come curve parametriche in Blender;
- esportazione punti di controllo come file di testo;
- lettura delle curve nell'algoritmo principale;
- deformazione e spostamento punti di controllo nello spazio tramite interpolazione trilineare;
- generazione facce dei tubi sviluppati lungo le curve;
- uso diretto o esportazione `.stl` del risultato.

3.6.3 Definizione tessere

Ogni tessera è definita tramite curve di Bézier multiple. Con questa tecnica, è possibile descrivere curve con più punti di controllo. Viene artificialmente imposta la continuità C^1 a scopo estetico tramite lo spostamento dei punti di controllo in corrispondenza delle giunzioni tra le singole curve. Esse sono tutte comprese nel cubo unitario, affinché vengano correttamente trasformate dall'interpolazione trilineare di cui alla Sezione 3.6.4. Una delle maglie definite è visibile in Figura 3.12.

Una volta definita ogni tessera in Blender, esse vengono esportate tramite Python in un file di testo per essere utilizzate dall'algoritmo principale. Lo script, anziché convertire in mesh le curve come elenco di vertici discretizzandole, mantiene la rappresentazione parametrica, salvando nel file le coordinate dei punti di controllo di ogni Bézier. Il beneficio di questa scelta è la possibilità di migliorare il raccordo tra le tessere come illustrato nella Sezione 3.6.4,

Il formato testuale segue una formattazione semplice così definita:

(a) (b)

Figura 3.12: Curve multi Bézier per realizzare le tessere in Blender; (a) insieme di tutte le tessere realizzate; (b) primo piano di una maglia bassa.

- ogni curva è separata dalla successiva da una linea vuota;
- per ogni punto di ciascuna curva, viene esportata una linea con le coordinate;
- di ogni punto di estremo della curva, sono esportate le coordinate dei due punti di controllo della derivata;
- i punti di controllo sono separati da un `|` o "pipe";
- le coordinate x, y, z sono salvate come floating point e separate da uno spazio.

Si riporta per completezza un estratto, qui espresso con numeri troncati alla terza cifra decimale per motivi di spazio, di due curve esportate dallo script Python in Blender.

```
-0.119 0.500 0.500|0.000 0.500 0.500|0.500 0.500 0.500
0.584 0.798 0.500|0.000 0.800 0.500|-0.295 0.800 0.500

-0.119 0.247 0.500|0.000 0.250 0.500|0.170 0.253 0.500
0.305 -0.012 0.186|0.418 0.129 0.500|0.427 0.139 0.522
0.407 0.166 0.604|0.385 0.207 0.644|0.363 0.248 0.685
0.339 0.303 0.683|0.337 0.371 0.540|0.337 0.398 0.484
0.257 0.595 0.417|0.244 0.675 0.497|0.222 0.815 0.637
0.251 0.857 0.449|0.250 1.000 0.449|0.246 1.295 0.449
```

Il ruolo di questo passaggio è di sfruttare gli strumenti di modellazione e visualizzazione di Blender nella definizione delle tessere, per poi preparare i dati della curva in un formato facilmente analizzabile dallo strumento software principale sviluppato in Rust.

3.6.4 Composizione tessere

Dopo aver definito tutte le possibili tessere, è necessario applicare una trasformazione per portarle alle coordinate destinazione, deformandole opportunamente.

Interpolazione trilineare

Le tessere delle maglie sono definite all'interno di un cubo di lato unitario, con un vertice nell'origine. Per creare una mesh completa a scopo di visualizzazione, è necessario trasformare ogni tessera, traslandola, posizionandola e opportunamente deformandola in base alle coordinate target calcolate in $\mathbb{R}^3$. I vertici del cubo visualizzato in *wireframe* in Figura 3.12 sono spostati alla loro posizione finale, come avviene per la rete di controllo di un *lattice*.

Dopodiché, al contenuto viene applicata una interpolazione trilineare. Essa consiste in una generalizzazione a tre dimensioni della interpolazione lineare e bilineare.

La prima interpola due punti in uno spazio $\mathbb{R}^n$ di dimensione n a piacere tramite un singolo parametro u. Definiti P_0 e P_1 i punti tra cui interpolare e un parametro $u \in [0, 1]$, il punto interpolato si ottiene come:

$$P(u) = P_0 \cdot (1 - u) + P_1 \cdot (u)$$

A sua volta il concetto può essere esteso a due parametri di controllo e quattro punti tra cui interpolare (bilineare):

$$P(u, v) = P_{0,0} \cdot (1 - u) \cdot (1 - v) + P_{1,0} \cdot u \cdot (1 - v) + P_{0,1} \cdot (1 - u) \cdot v + P_{1,1} \cdot u \cdot v$$

In ultimo, nella trasformazione dei tile viene adoperata la versione trilineare, con tre parametri di controllo (u, v, w) e otto punti tra cui interpolare. Essa può essere trattata come una interpolazione lineare controllata da w, operata sui risultati di due interpolazioni bilineari sulle facce opposte, controllate da (u, v). Di conseguenza:

$$\begin{aligned} P(u, v, w) &= P_0(u, v) \cdot (1 - w) + P_1(u, v) \cdot w \\ P(u, v, w) &= P_{0,0,0} \cdot (1 - u) \cdot (1 - v) \cdot (1 - w) + P_{1,0,0} \cdot u \cdot (1 - v) \cdot (1 - w) \\ &\quad + P_{0,1,0} \cdot (1 - u) \cdot v \cdot (1 - w) + P_{1,1,0} \cdot u \cdot v \cdot (1 - w) \\ &\quad + P_{0,0,1} \cdot (1 - u) \cdot (1 - v) \cdot w + P_{1,0,1} \cdot u \cdot (1 - v) \cdot w \\ &\quad + P_{0,1,1} \cdot (1 - u) \cdot v \cdot w + P_{1,1,1} \cdot u \cdot v \cdot w \end{aligned} \tag{3.1}$$

Correzione derivata prima

Si ricorda dapprima l'obiettivo, ossia ottenere i vertici che definiscano una mesh poligonale da potere visualizzare in un programma qualsiasi. Come struttura, il filo in ogni tessera è definito come una sequenza di curve di Bézier collegate in modo da preservare la continuità delle derivate e mantenere la curva complessiva all'interno della classe C^1 e il corretto diametro del filo dopo alla trasformazione.

Un approccio *naïve*, computazionalmente più rapido ma esteticamente meno accurato di applicare la trasformazione consiste nel generare una sola volta i vertici della mesh per il cubo unitario. Dopodiché, si applica la interpolazione trilineare direttamente a ogni vertice ottenuto, per portarlo alla posizione corretta. Seppur semplice da implementare, ciò porta a schiacciamenti visivi del filo, che diviene a base ellittica anziché circolare.

Per migliorare la visualizzazione, la trasformazione viene invece applicata ai punti di controllo delle curve parametriche. Dopo aver spostato le curve parametriche nello spazio, vengono generati dei nuovi vertici, dato il diametro.

Questo approccio funziona senza modifiche sui punti interni che definiscono il filo. Tuttavia, i punti di controllo degli estremi, dove il filo sarà collegato alle altre tessere, necessitano di una ulteriore correzione. L'obiettivo è mantenere la continuità C^1 non solo all'interno della stessa maglia, bensì anche quando esse sono poste in sequenza.

Si desidera che il punto di controllo imponga una derivata con direzione normale alla superficie di contatto con le altre tessere, in questo caso rispetto alla faccia in basso. La banale trasformazione del punto di controllo non garantisce la ortogonalità con la faccia di confine a cui si riferisce. Per migliorarne il comportamento, si segue questo procedimento. Definiti il punto sulla faccia di confine P_0, il suo punto di controllo della derivata P_c appena trasformato e il versore $\hat{N}$ normale alla faccia di confine di interesse, si ottiene:

$$P_{corretto} = P_0 + \langle P_c - P_0, \hat{N} \rangle$$

A sua volta, il versore $\hat{N}$ necessario al calcolo è ottenibile come prodotto vettoriale di due lati del cubo trasformati con un punto in comune. Come esempio per i lati inferiori del cubo unitario, opportunamente trasformati:

$$\begin{aligned} \vec{a} &= P(1,0,0) - P(0,0,0) \\ \vec{b} &= P(0,1,0) - P(0,0,0) \\ \hat{N} &= norm(\vec{a} \times \vec{b}) \end{aligned} \tag{3.2}$$

Versione verosimile

Applicando tutti i passaggi precedentemente descritti, si ottiene una pipeline per comporre una mesh poligonale completa e verosimile, che visualizzi ogni maglia con la struttura del filo corrispondente e tridimensionale.

In questa tesi, i vertici calcolati in Rust vengono mostrati in Godot Engine, da cui si può liberamente esportare la mesh in formato `.stl`, compatibile con tutti i software di modellazione 3D poligonale. Per ottenere la visualizzazione in Figura 3.13b, il file `.stl` è stato importato in Blender tramite un componente aggiuntivo di Godot [22]. Posizionata una camera e impostato un materiale base adatto, è stato effettuato il rendering con il motore Cycles. Grazie alle correzioni prima operate, non sono esteticamente visibili giunture tra le maglie interpolate, e il risultato appare come organizzato in una struttura unica.

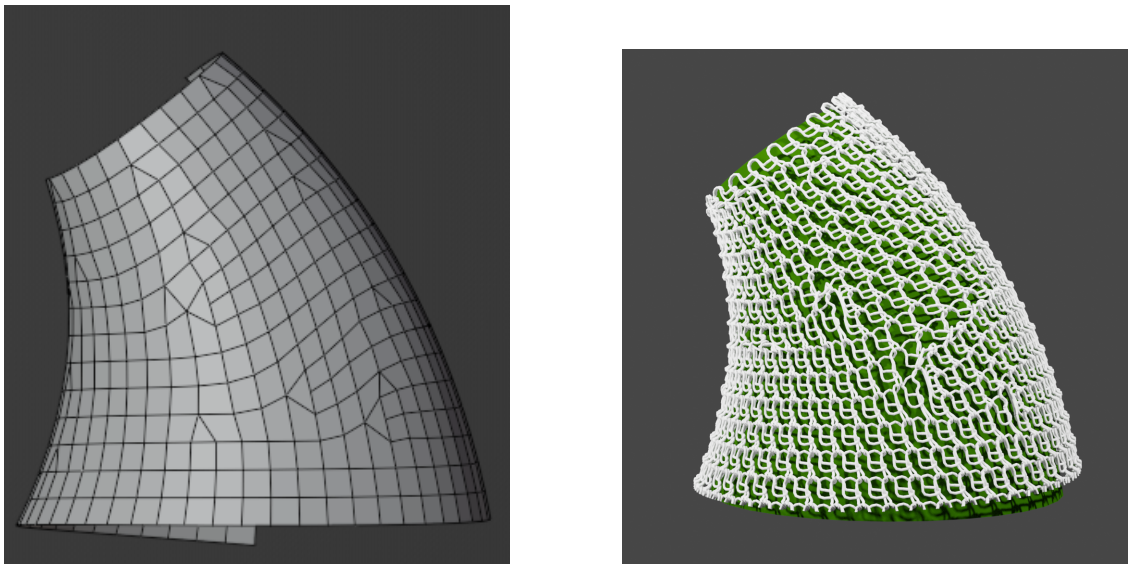
Versione schematica

È possibile visualizzare il risultato tramite semplici facce, in versione schematizzata, a cui è possibile opzionalmente applicare una texture. Ciò riduce la fedeltà visiva del risultato, a fronte di un miglioramento dei tempi di calcolo.

Il prodotto ottenuto, riportato in Figura 3.13a, è utile per una visualizzazione in wireframe come era stata ottenuta da Çapunaman et al. [1]. Essa verrà utilizzata per confrontare l'organizzazione delle maglie con il lavoro di riferimento.

Nel caso semplice di una maglia con quattro lati, i punti trasformati tra cui costruire il quadrilatero saranno banalmente:

$$P(0,0,\frac{1}{2}), P(0,1,\frac{1}{2}), P(1,1,\frac{1}{2}), P(1,0,\frac{1}{2})$$



(a) (b)

Figura 3.13: Confronto delle visualizzazioni supportate; (a) versione schematica; (b) rendering della versione verosimile.

Il valore $\frac{1}{2}$ pone il piano generato a metà del cubo unitario dopo la trasformazione. Le coordinate u e v così individuate si trovano in corrispondenza dei lati del cubo trasformato. Ogni quadrilatero è suddiviso in due triangoli. Anch'essi sono percorsi in senso orario, per rispettare il *winding order* di Godot.

Capitolo 4

Risultati

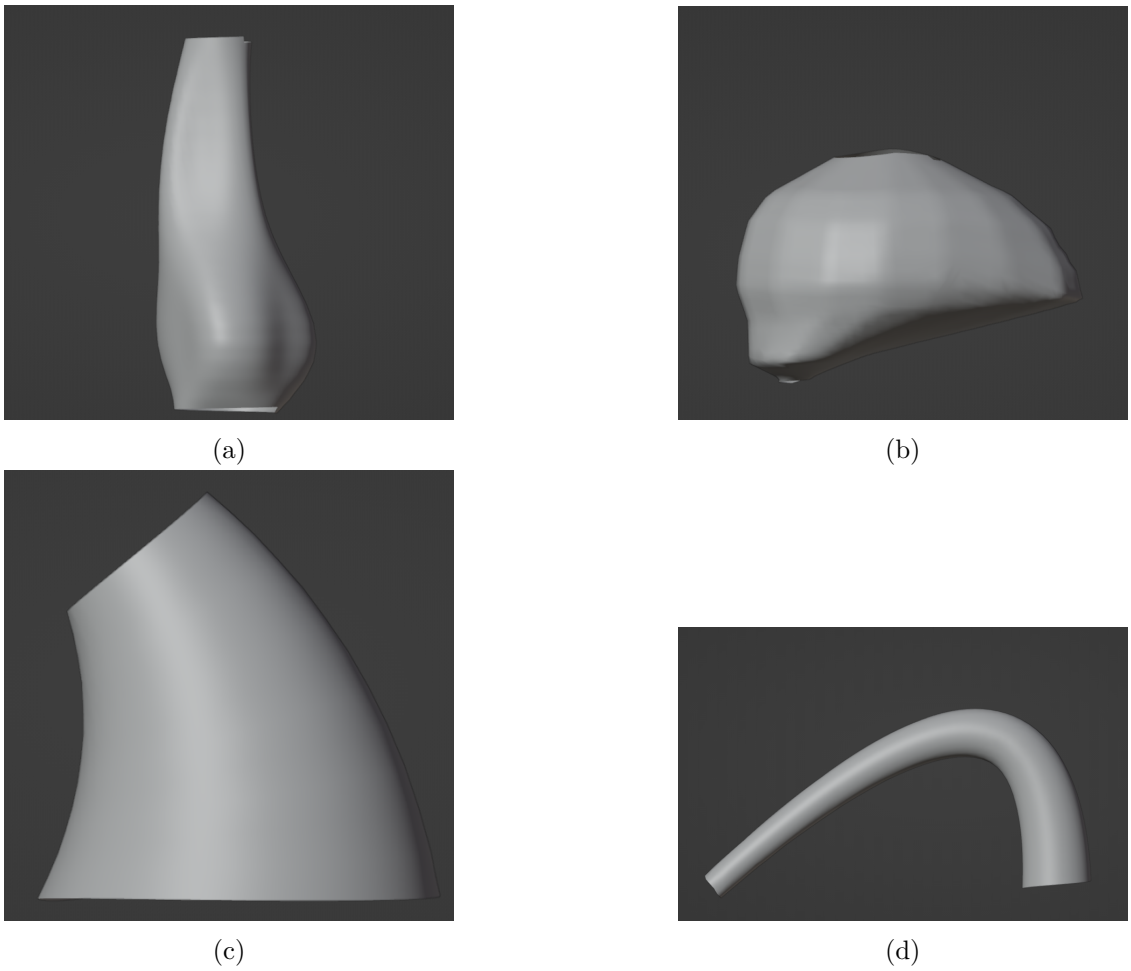
Implementati entrambi gli algoritmi affinché possano essere confrontati, essi vengono eseguiti su alcune superfici utilizzate nel lavoro originale. Si è tentato di impostare le dimensioni della maglia più simili possibili ai valori utilizzati nel lavoro di riferimento, processo tuttavia frutto di tentativi valutati visivamente, poiché i valori esatti non sono stati riportati.

Si ringrazia Özgüç Çapunaman per aver messo a disposizione il file Rhinoceros con alcune delle superfici di esempio per poter replicare i risultati. Per estrarre la superficie da Rhinoceros, essa è stata suddivisa manualmente tramite lo strumento "isocurve". Queste sono state esportate in formato `.obj`, per poter essere aperte con Blender e adattate tramite uno script di conversione all'uso nel software sviluppato in Rust. In aggiunta, sono definite alcune altre superfici utili a mettere in evidenza le peculiarità degli algoritmi comparati.

I test verranno valutati sulle superfici A, B, C e D mostrate in Figura 4.1, di cui le prime due sono estratte dal lavoro di riferimento e le altre create *ad hoc*. Esse sono analizzate per approfondire, rispettivamente, il comportamento conseguente dalla presenza delle seguenti proprietà:

- forma pressoché cilindrica con piccole variazioni asimmetriche del raggio;
- aumenti ingenti del raggio della superficie in modo asimmetrico, pur mantenendo la direzione della curva di controllo costante;
- piccola variazione della direzione della curva di controllo;
- ingente variazione della direzione della curva di controllo.

Le prime due proprietà servono a verificare la capacità di gestire variazioni asimmetriche del raggio della superficie, poiché il supporto a questa caratteristica coincide con l'obiettivo primario del lavoro di riferimento. Secondariamente, poiché le prime due superfici di test considerate si sviluppano seguendo una direzione rigidamente verticale, si esplora il comportamento degli algoritmi nel caso essa si modifichi, prima con una variazione della direzione minuta, successivamente nel caso di una variazione ingente.



(a) (b)

(c) (d)

Figura 4.1: Superfici prese in considerazione nei test; (a) superficie A, da lavoro di riferimento; (b) superficie B, da lavoro di riferimento; (c) superficie C, piccole variazioni della direzione; (d) superficie D, con ingenti variazioni della direzione della curva di controllo.

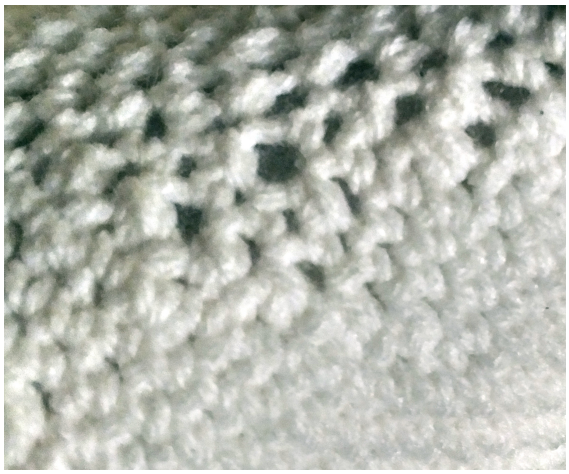


Figura 4.2: Aspetto visivo delle zone di tessuto con dimensione delle maglie vicina a quella target (in basso) o stirate (in alto).

4.1 Definizione delle metriche valutate

I risultati sono stati valutati tramite i seguenti metodi:

- distribuzione dell'errore percentuale delle lunghezze degli archi [11];
- skew delle facce [23];
- tempi di calcolo;
- confronto visuale.

4.1.1 Errore percentuale della lunghezza degli archi

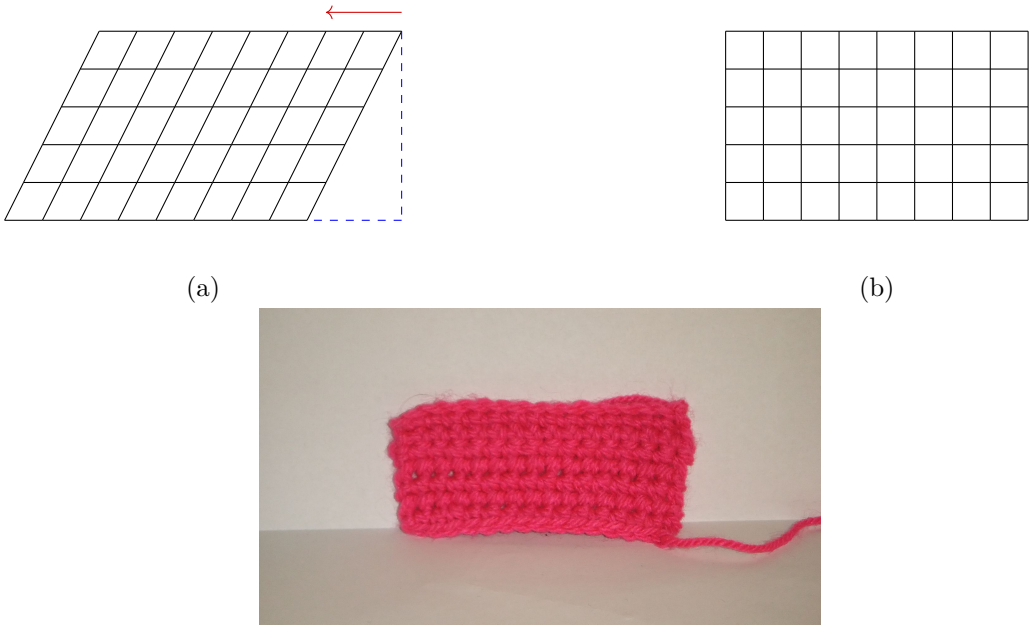
Si ricorda che il primo passo dell'algoritmo prevede l'impostazione della larghezza e dell'altezza delle maglie desiderata, come determinate con il metodo introdotto nella Sezione 1.2.1. Ciò è innanzitutto necessario affinché le dimensioni dell'oggetto calcolato siano simili a quelle reali. Inoltre, poiché le maglie hanno una limitata capacità di comprimersi o estendersi in modo elastico senza causare deformazioni visibili del tessuto, è di interesse verificare che la larghezza e l'altezza delle maglie simulate dall'algoritmo siano simili a quelle ideali. In Figura 4.2 sono mostrate in basso alcune maglie vicine alla propria dimensione a riposo e in alto altre sottoposte invece a forze di trazione da cui sono forzate ad assumere una lunghezza maggiore di quella target.

Allo scopo di valutare la proprietà poc'anzi delineata, il lavoro di Narayanan et al. [11] propone l'uso dell'errore percentuale della lunghezza degli archi.

Si definisce l'errore percentuale E_p come segue:

$$E_p = \frac{\|P_A - P_B\| - l}{l}$$

In cui:



(a) (b)

(c)

Figura 4.3: Confronto struttura calcolata e reale in presenza di skew ingente; (a) reticolo calcolato; (b) posizione di riposo del reticolo; (c) comportamento reale del cucito.

- P_A e P_B sono le coordinate dei due vertici che delimitano l'arco in analisi;
- l è la lunghezza target;
- $\|\cdot\|$ è l'operatore di norma euclidea.

Più questo indice, per ogni arco, ha risultati vicini a zero, meno stiramenti e compressioni esistono nella struttura cucita. Di conseguenza, è di interesse analizzarne la distribuzione includendo tutti gli archi, per poi confrontare media e varianza. L'obiettivo è ottenere media e varianza molto vicini a zero.

4.1.2 Deformazione *skew*

In seguito, viene misurata una metrica di skew. Essa, oltre a rappresentare un metodo di valutazione della qualità delle mesh utilizzato in software di computer grafica esistenti [24], ha una valenza ulteriore nell'ambito di analisi di questa tesi, a causa del comportamento mostrato in Figura 4.3: le facce che l'algoritmo genera con uno skew ingente perderanno questa qualità una volta cucite in un oggetto reale, a causa delle forze di trazione ed elastiche del filo. La Figura 4.3c mostra che il tessuto perde lo skew e le maglie tendono a tornare a sovrapporsi verticalmente. Ciò può contribuire a gibbosità antiestetiche nell'oggetto finito quando vengono affiancate zone con basso skew, che rimangono fedeli a quanto calcolato, e zone con skew ingente, le quali applicano delle forze di deformazione non previste dagli algoritmi.

Figura 4.4: Definizione dell'angolo α dai segmenti mediani; (a) disposizione dei segmenti mediani su un quadrilatero; (b) disposizione dei segmenti mediani su un triangolo.

Lo skew per ogni faccia è definito come il $|cos(\alpha)|$, in cui α è l'angolo tra i segmenti mediani del poligono, esplicato nella Figura 4.4. La metrica così definita ha il pregio di essere applicabile indipendentemente, senza modifiche, sia su quadrilateri, sia su facce triangolari [23].

Viene analizzata la distribuzione dei valori di skew: una maggiore concentrazione attorno a 0 indica una mesh più regolare. Poiché i due algoritmi sulla stessa superficie e con le stesse dimensioni di maglia generano un numero diverso di punti, gli istogrammi vengono normalizzati in altezza in modo che la somma dei valori sia 1, affinché possano essere confrontati direttamente.

4.1.3 Tempi di calcolo

Poiché lo strumento software sviluppato può essere adoperato in contesti di virtual prototyping in cui è utile godere di un'anteprima rapida, vengono analizzati i tempi di calcolo in diverse configurazioni. A parità di superficie, è possibile modificare le dimensioni impostate (larghezza e altezza) delle singole maglie, per esempio al variare del diametro del filo. Ciò produce un ammontare di vertici dipendente da esso; di conseguenza, si valuta il comportamento del tempo di calcolo in funzione del numero di vertici generati. Oltre al valore di tempo assoluto necessario per generare il grafo, data la dimensione della maglia, viene riportato il numero di maglie al secondo che l'algoritmo è in grado di generare.

Il calcolo del tempo impiegato avviene tramite la struttura `Instant` inclusa nelle librerie standard di Rust [25], la quale assicura che gli istanti di tempo da essa estratti seguano un orologio interno i cui valori restituiti siano monotoni strettamente crescenti. Vengono salvati due oggetti `Instant` all'inizio e alla fine del calcolo, poi confrontati tramite il metodo `duration_since(&self, earlier: Instant)`, che restituisce il risultato all'interno di una struttura `Duration`.

Si riporta un estratto del codice utilizzato per il calcolo e la stampa del tempo impiegato in millisecondi.

```
let start = Instant::now();
// Inizializzazione grafo
let mut gr: Box<dyn KnittableGraph> = Box::new(
    nodes::CrochetGraph::new(surf.clone())
);
// Effettivo calcolo delle maglie
gr.generate_graph();
```

```
let end = Instant::now();

// Si ottiene l'oggetto Duration e lo si converte in millisecondi
let time: Duration = end.duration_since(start);
println!("{}", time.as_millis());
```

4.1.4 Confronto visuale

Infine, viene operato un confronto visuale che include la realizzazione pratica cucita di alcune delle superfici oggetto di analisi. Da questa valutazione si può verificare l'impatto strutturale delle short row sul prodotto finito.

4.2 Risultati sperimentali

All'intero di questa sezione sono raccolti e confrontati i risultati ottenuti attraverso le metriche appena definite.

4.2.1 Metriche di regolarità

Questa sezione riporta il confronto delle metriche di errore percentuale della lunghezza degli archi e della deformazione di skew su tutte le superfici in analisi.

Superficie A

Si riporta innanzitutto un confronto visuale nella Figura 4.5. Sono mostrati, da sinistra a destra:

- la superficie target, estratta in isocurve da Rhinoceros;
- la spirale calcolata;
- il risultato dell'algoritmo di riferimento;
- il risultato dell'algoritmo presentato in questa tesi.

Data la superficie, la spirale generata è comune a entrambi e dipende dalla altezza della maglia impostata.

I test dell'errore percentuale, eseguiti sulla prima superficie fornita dal lavoro di riferimento, sono riportati in Figura 4.6. L'algoritmo presentato in questa tesi presenta dei valori comparabili all'altro, sullo stesso ordine di grandezza, come evidente rispettivamente dalle Figure 4.6f e 4.6e. La media è a vantaggio di quello proposto, la varianza è sostanzialmente identica. Si denota un comportamento speculare tra gli archi di tipo `Course` e `Wale`. Infatti, poiché l'algoritmo di riferimento campiona direttamente ogni giro della spirale tramite la dimensione della maglia impostata, giova di un ottimo contributo sugli archi `Course` visibile nell'istogramma in Figura 4.6a a discapito dell'accuratezza negli archi di tipo `Wale`, come evidente in Figura 4.6b. D'altro canto, l'algoritmo proposto in questa tesi presenta valori ottimali per gli archi di tipo `Wale` riportati in Figura 4.6d, ma meno accurati per quelli di tipo `Course` in Figura 4.6c.

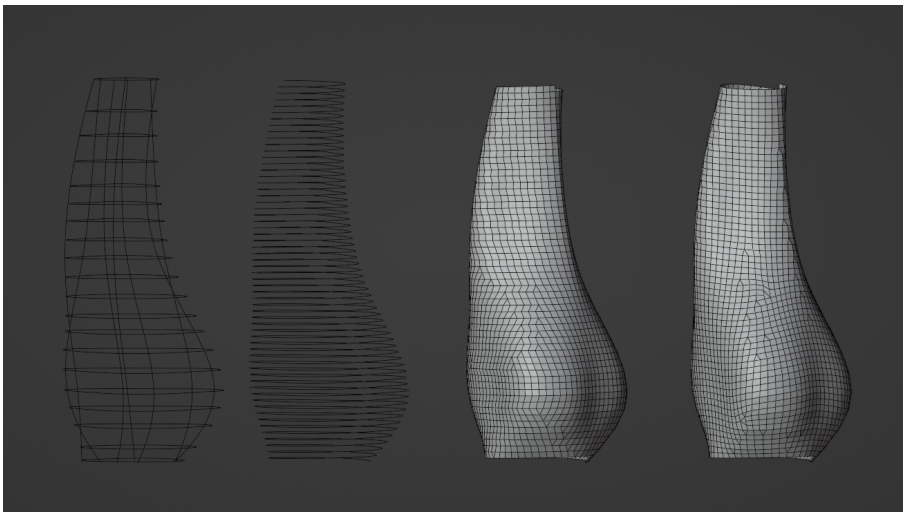


Figura 4.5: Confronto risultati eseguiti sulla superficie A

Per quanto concerne la metrica di skew, si evince che l'algoritmo di riferimento in questa superficie genera, mediamente, deformazioni maggiori, comportamento anche rilevabile visivamente. L'istogramma in Figura 4.7a è meno concentrato attorno a 0 e appaiono più facce con valori alti di deformazione. Invece, quello riportato in Figura 4.7b relativo all'algoritmo proposto presenta valori più concentrati intorno a zero, una media più vicina a zero e anche minore varianza.

Si denota anche una differenza nella distribuzione della deformazione di skew sulla superficie. Nell'algoritmo di riferimento, per costruzione, le facce di diminuzione e quelle con maggiore deformazione sono collocate a coordinate vicine a $v = 0.5$ e sono assenti per valori prossimi a $v = 0$; viceversa, nella disposizione dell'algoritmo proposto esse sono distribuite più uniformemente nell'intervallo $v \in [0, 1]$.

Per completezza, poiché come in Figura 4.8 è possibile variare la dimensione della maglia impostata, ci si potrebbe chiedere se l'esito delle metriche dipenda direttamente dal valore di questo parametro. Si riporta in Figura 4.9 l'andamento delle due metriche in funzione di esso. Si desume che la differenza della media di E_p tra i due algoritmi si riduce con la diminuzione della dimensione della maglia impostata senza mai azzerarsi, mentre la differenza della deformazione di skew rimane costante con piccole oscillazioni. Non è possibile ridurre ulteriormente la dimensione della maglia impostata senza rilevare errori, causati dall'aumento smisurato del numero di punti generati, combinato a un eccessivo incremento dei tempi di calcolo.

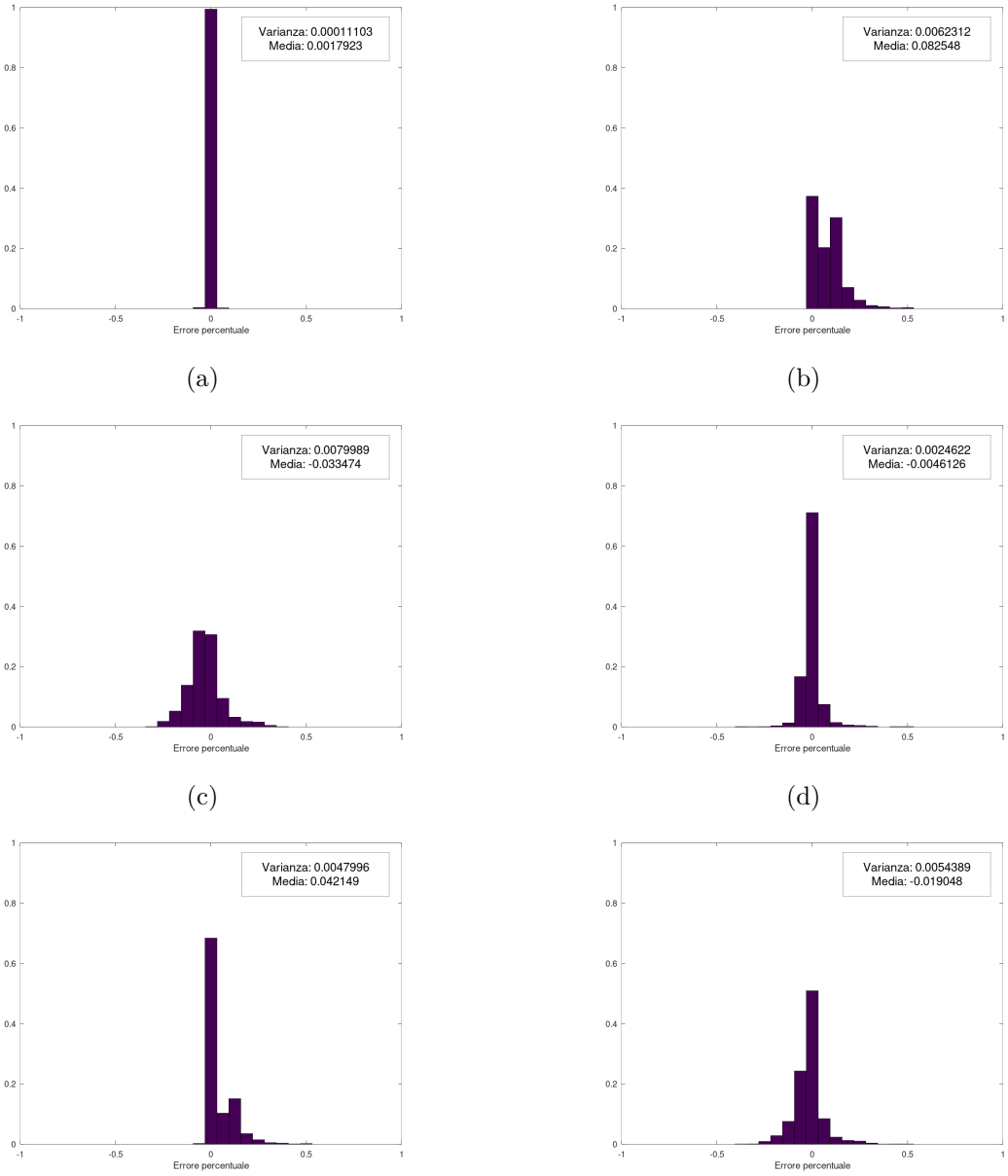
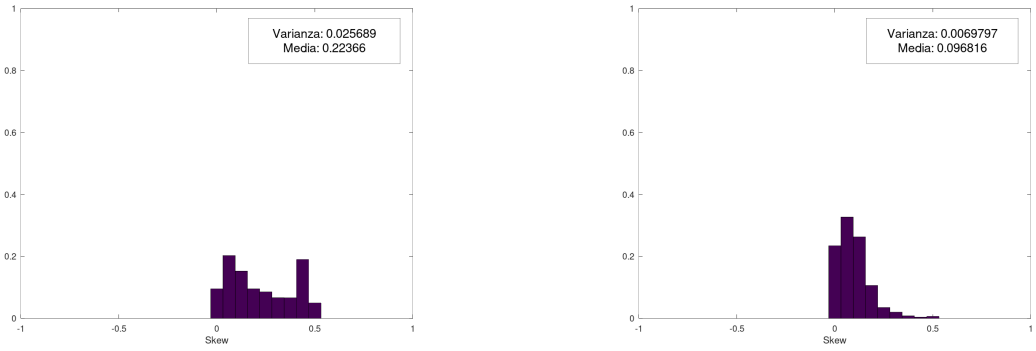
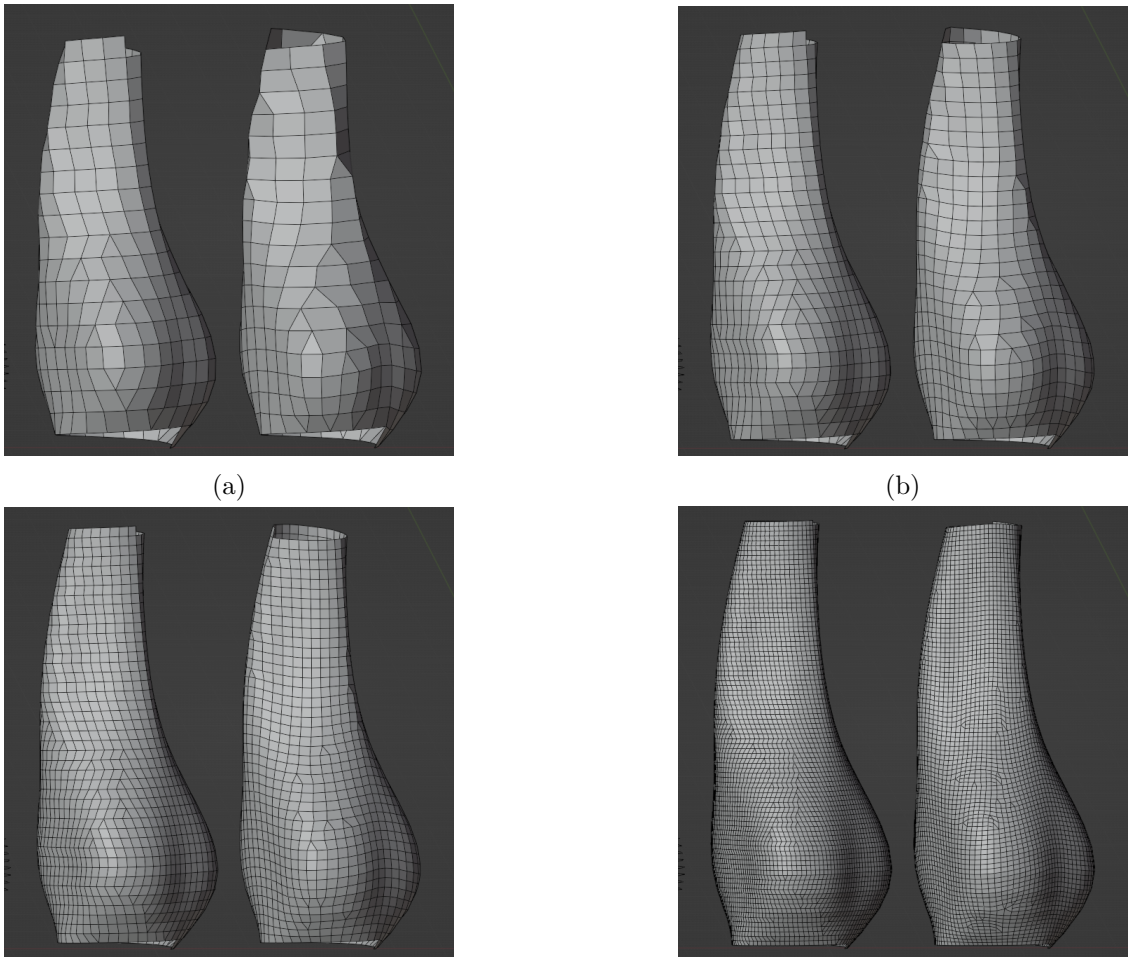


Figura 4.6: Istogrammi normalizzati dell'errore percentuale della lunghezza degli archi; (a,b) risultati dell'algoritmo di riferimento, rispettivamente per archi `Course`, `Wale`; (c,d) risultati dell'algoritmo proposto, rispettivamente per archi `Course`, `Wale`; (e,f) risultati calcolati includendo tutti gli archi, rispettivamente per l'algoritmo di riferimento e quello proposto.



(a) (b)

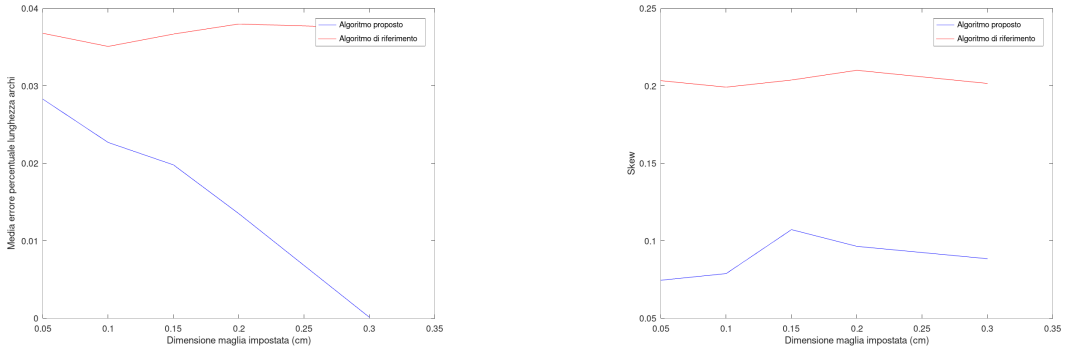
Figura 4.7: Distribuzione e confronto distribuzione skew; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.



(a) (b)

(c) (d)

Figura 4.8: Superficie A generata con diverse dimensioni W della maglia; (a) $W = 0.45cm$; (b) $W = 0.3cm$; (c) $W = 0.2cm$; (d) $W = 0.1cm$.



(a) (b)

Figura 4.9: Andamento delle metriche in funzione della dimensione della maglia impostata; (a) media errore percentuale della lunghezza degli archi; (b) media della deformazione skew.

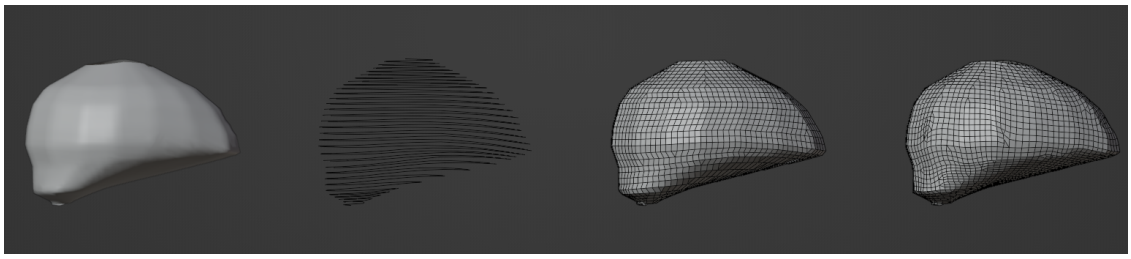
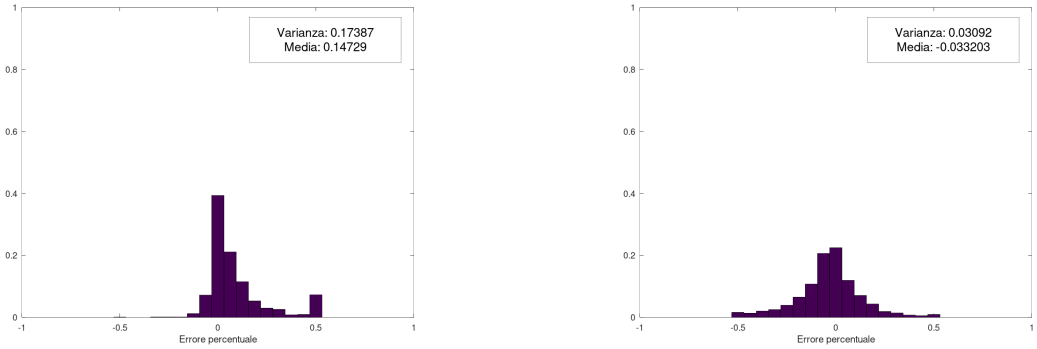


Figura 4.10: Comparazione visiva dei risultati sulla superficie B; in ordine da sinistra a destra: oggetto B, spirale, esito algoritmo di riferimento, esito algoritmo proposto.



(a) (b)

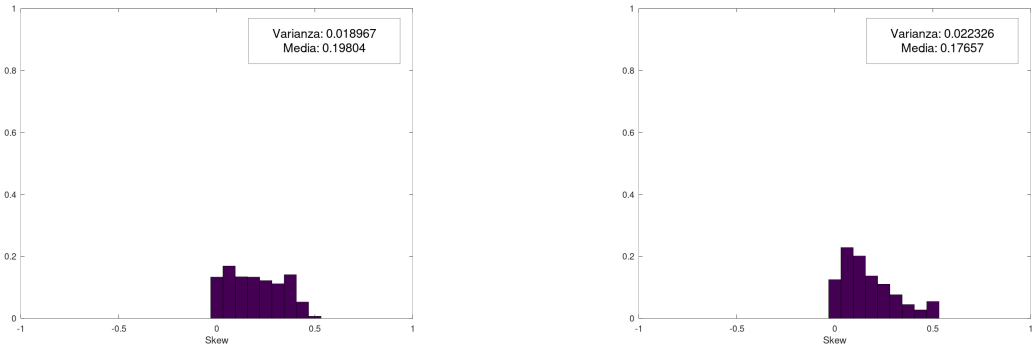
Figura 4.11: Istogrammi normalizzati dell'errore percentuale della lunghezza degli archi sulla superficie B; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.

Superficie B

La seconda superficie, anch'essa estratta dal lavoro di riferimento, è caratterizzata da una direzione di sviluppo costantemente verticale combinata a una variazione ingente e asimmetrica del raggio della superficie.

Innanzitutto in Figura 4.10 si nota che, a causa dell'ingente variazione asimmetrica del raggio, l'algoritmo proposto inserisce delle short row nella zona inferiore destra dell'oggetto in analisi. In questa zona, l'algoritmo di riferimento produce degli archi molto più lunghi dell'altezza desiderata; ciò è evidente nel picco presente in corrispondenza di 0.5 in Figura 4.11a. Poiché la differenza di comportamento nei grafici dei soli archi `Course` e `Wale` è osservabile in tutte le superfici, si riportano in Figura 4.11 solo gli istogrammi relativi a tutti gli archi.

Accanto al risultato migliorativo della metrica primaria, si assiste a una equipollenza della deformazione di skew: come illustrato in Figura 4.12, l'istogramma in Figura 4.12b è minimamente più concentrato intorno a zero, con valori di media e varianza sullo stesso ordine di grandezza.



(a) (b)

Figura 4.12: Distribuzione e confronto distribuzione skew della superficie B; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.

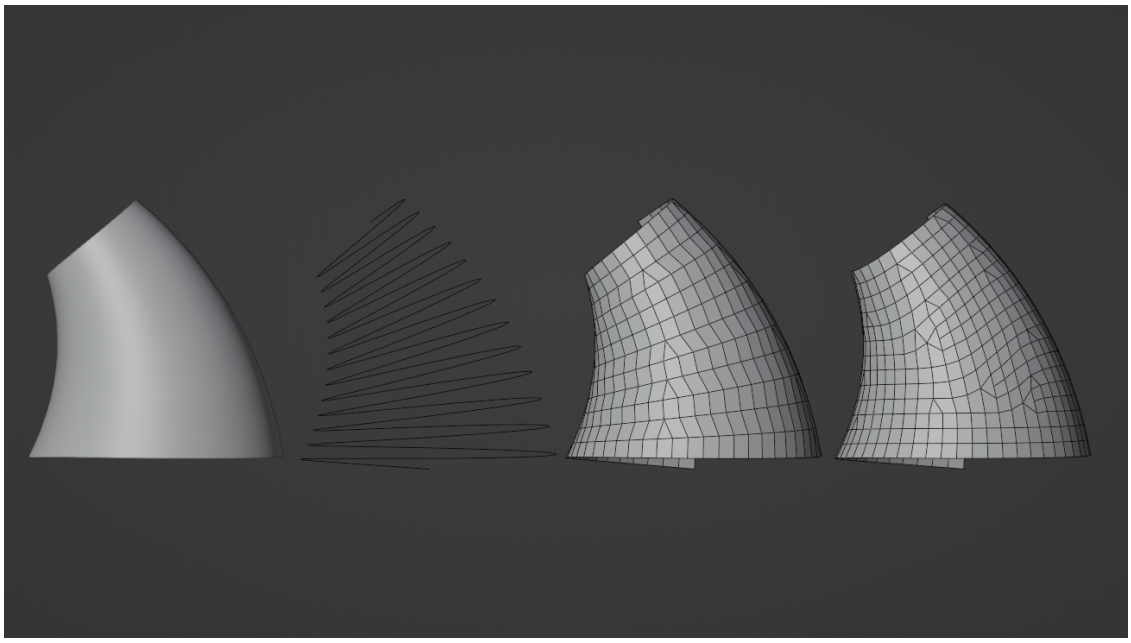


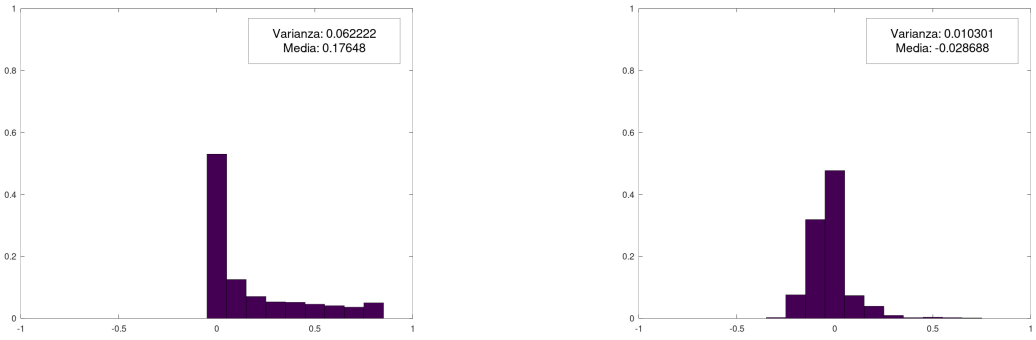
Figura 4.13: Comparazione visiva dei risultati sulla superficie C; in ordine da sinistra a destra: oggetto C, spirale, esito algoritmo di riferimento, esito algoritmo proposto.

Superficie C

La terza superficie in analisi, simile a un tronco di cono, integra una minima variazione della direzione della curva di controllo. In Figura 4.13 si riporta un confronto visivo del risultato: nonostante il lieve cambio di direzione, è possibile rilevare la corretta aggiunta di short row correttive sul lato destro dell'oggetto.

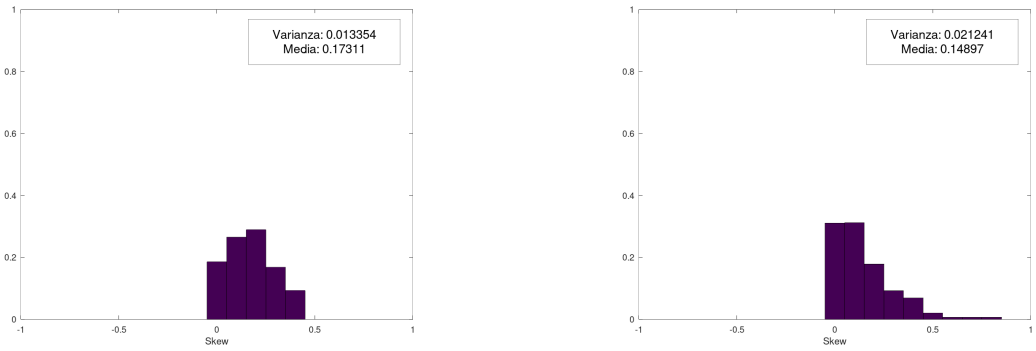
I risultati della distribuzione dell'errore percentuale, riportati in Figura 4.14, mostrano una maggiore concentrazione attorno a zero degli istogrammi riscontrabile anche visivamente, che numericamente corrisponde a un miglioramento di 0.15 della media e di 0.05 della varianza.

Per quanto concerne la metrica di skew, come esposto in Figura 4.15, si assiste a un risultato simile tra i due algoritmi, con un leggero miglioramento della media e un minuto peggioramento della varianza.



(a) (b)

Figura 4.14: Istogrammi normalizzati dell'errore percentuale della lunghezza degli archi sulla superficie C; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.



(a) (b)

Figura 4.15: Distribuzione e confronto distribuzione skew della superficie C; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.

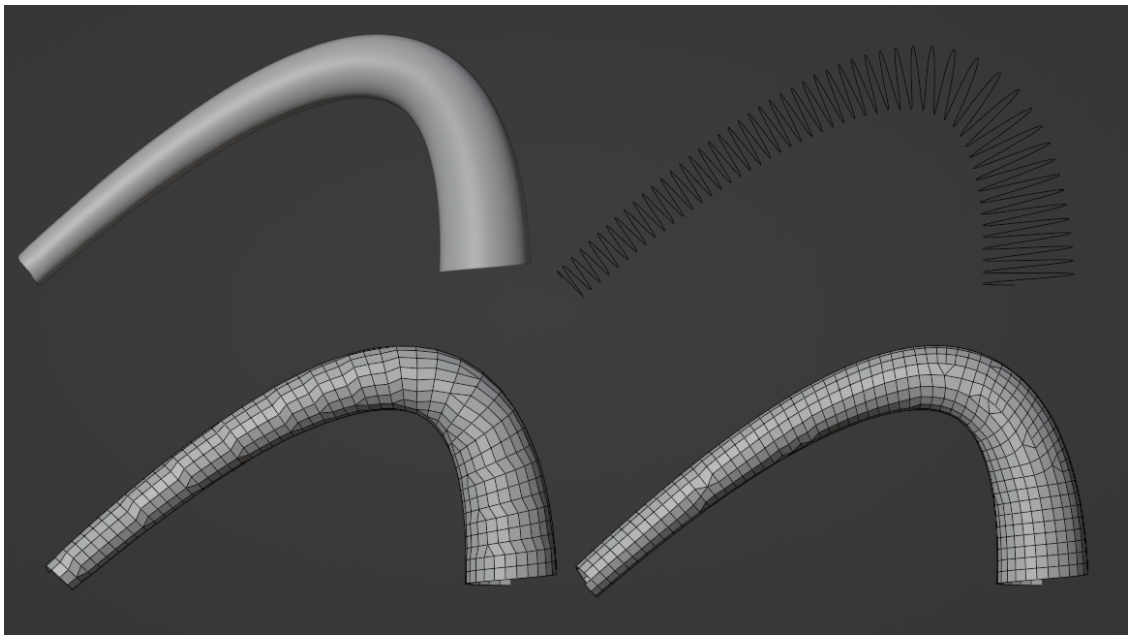


Figura 4.16: Comparazione visiva dei risultati sulla superficie D; in ordine da sinistra a destra, dall'alto verso il basso: oggetto D, spirale, esito algoritmo di riferimento, esito algoritmo proposto.

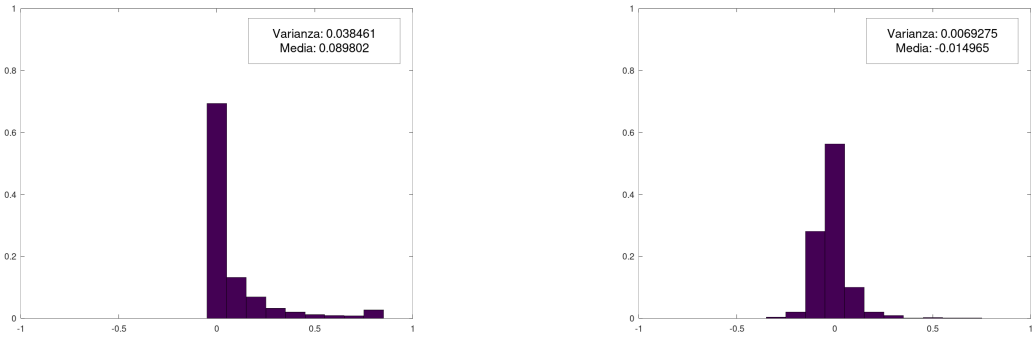
Superficie D

L'ultima superficie in analisi denota un ingente cambio di direzione di sviluppo della superficie combinato con un raggio del tronco di cono molto ridotto in dimensione rispetto a quello impostato nella superficie C.

Anche in questa superficie è vitale l'inserimento di short row correttive.

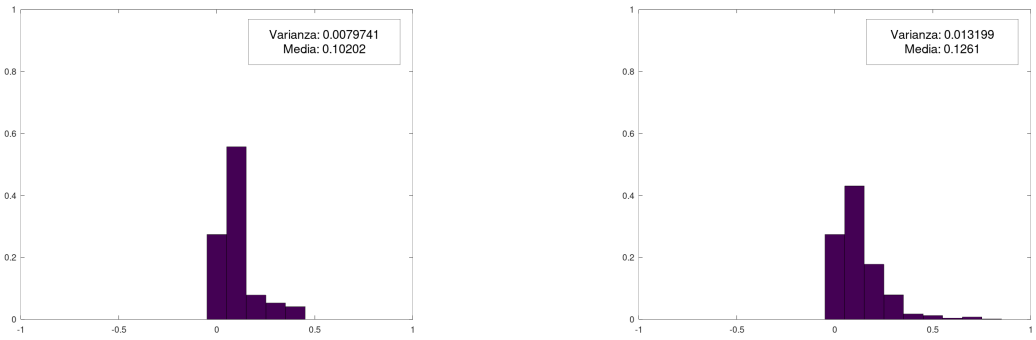
Per ciò che concerne i grafici in Figura 4.17, è di interesse notare il picco visibile in Figura 4.17a nella barra in corrispondenza di 0.8, causata dagli archi eccessivamente lunghi posti in direzione di $-\hat{N}(t)$ nel sistema di riferimento della curva di controllo.

Sulla metrica di skew, risulta leggermente migliore l'algoritmo di riferimento, con una differenza di 0.05 riguardo la media e circa 0.01 per la varianza.



(a) (b)

Figura 4.17: Istogrammi normalizzati dell'errore percentuale della lunghezza degli archi sulla superficie D; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.



(a) (b)

Figura 4.18: Distribuzione e confronto distribuzione skew della superficie D; (a) algoritmo di riferimento; (b) algoritmo proposto in questa tesi.

Superficie	**Differenza E_p**		**Differenza skew**	
	Media	**Varianza**	**Media**	**Varianza**
A	-0.02	0.00	-0.13	-0.02
B	-0.11	-0.14	-0.02	0.00
C	-0.15	-0.05	-0.02	$+0.01$
D	-0.07	-0.03	$+0.02$	$+0.01$

Tabella 4.1: Confronto della differenza di media e varianza delle metriche tra i risultati dei due algoritmi eseguiti su tutte le superfici oggetto di analisi

Superficie	P-value di E_p	P-value dello skew
A	2.2e-16	2.2e-16
B	2.2e-16	4.88e-07
C	2.2e-16	2.34e-03
D	2.2e-16	4.96e-07

Tabella 4.2: Esito p-value dei t-test.

Comparazione numerica dei risultati

La Tabella 4.1 riassume i risultati sperimentali: essa riporta la differenza dei valori di media e varianza ottenuti rispettivamente per la metrica E_p e per la deformazione di skew, confrontando tutte le quattro superfici in analisi. Un risultato minore di zero indica un risultato a favore dell'algoritmo proposto in questa tesi, uno maggiore di zero indica un esito a sfavore di esso e a vantaggio dell'algoritmo di riferimento.

La media dell'errore percentuale della lunghezza degli archi mostra un netto miglioramento nell'algoritmo proposto in questa tesi, rispetto a quello di riferimento, consistente in tutti gli oggetti osservati, in particolare in B e C. La varianza assume valori sullo stesso ordine di grandezza tra i due algoritmi in A e D, leggermente più alta per C e risultando notevolmente migliorata in B.

Per quanto concerne la metrica di deformazione skew, la sua media assume risultati comparabili in B, C e D, accostati a ottimi risultati a favore dell'algoritmo proposto in A. La sua varianza, invece, si modifica in modo trascurabile per tutte le superfici in analisi.

Ciò conferma l'ipotesi iniziale che gli algoritmi tendano a sovrapporsi per superfici che di discostano poco da una forma cilindrica, quale la A, e divergano in presenza di cambi di direzione della curva di controllo come in C e D o di espansioni trasversali asimmetriche come in B.

Secondariamente, allo scopo di validare dal punto di vista statistico le differenze rilevate, si procede al calcolo del t-test tramite R e ne si riporta il p-value conseguente nella Tabella 4.2. Poiché tutti i valori rilevati sono di molto minori della soglia 0.05, si conclude che i risultati dei due algoritmi confrontati sono statisticamente differenti.

4.2.2 Analisi prestazionale

In questa sezione si analizzano i tempi di calcolo, misurati tramite il metodo illustrato in Sezione 4.1.3.

I test sono stati eseguiti su un calcolatore con sistema operativo GNU/Linux, distribuzione Ubuntu 20.04 LTS "Focal Fossa". Per completezza, si riportano le caratteristiche del processore estratte tramite il comando `lscpu`.

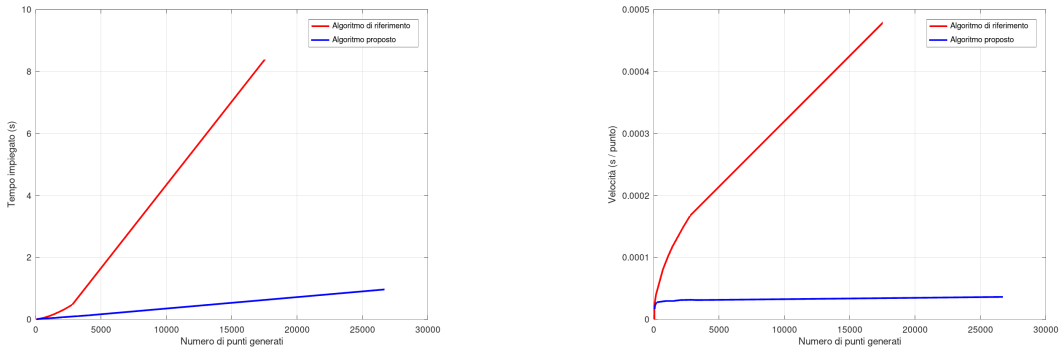
```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          24
On-line CPU(s) list:             0-23
Thread(s) per core:              2
Core(s) per socket:              12
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           113
Model name:                      AMD Ryzen 9 3900X 12-Core Processor
Stepping:                        0
Frequency boost:                 enabled
CPU MHz:                         2200.000
CPU max MHz:                     4672,0698
CPU min MHz:                     2200,0000
BogoMIPS:                        7600.01
Virtualization:                  AMD-V
L1d cache:                       384 KiB
L1i cache:                       384 KiB
L2 cache:                        6 MiB
L3 cache:                        64 MiB
NUMA node0 CPU(s):               0-23
```

Prendendo ad esempio la superficie C precedentemente mostrata in Figura 4.13, i risultati in Figura 4.19 mostrano risultati promettenti.

Sapendo che gli algoritmi possono generare un numero di maglie leggermente diverso, per completezza è stato normalizzato il tempo di calcolo rispetto al numero totale di maglie prodotte.

L'algoritmo di riferimento manifesta prestazioni comparabili per un numero basso di nodi generati. Tuttavia, come evidente in Figura 4.19a, all'aumentare del numero di nodi generati l'algoritmo di riferimento non è in grado di scalare in maniera efficiente e diviene subito più lento. Ciò è dovuto al meccanismo di collegamento tra i vertici in rige vicine: per ogni punto di ogni riga, viene verificato quale dei punti della riga successiva sia più vicino ad esso. Da ciò si desume che, date due righe di N punti, verranno calcolate NxN distanze geodetiche. L'ottimizzazione dell'algoritmo tramite la limitazione del numero di confronti da calcolare può essere oggetto di studi futuri.

Viceversa, l'algoritmo proposto in questa tesi mantiene un comportamento pressoché lineare in funzione del numero di nodi generati. Infatti, nonostante venga introdotto un piccolo *overhead* a causa delle logiche per integrare le short row, compreso il meccanismo



(a) (b)

Figura 4.19: Tempi di calcolo in s in funzione del numero di nodi generati

di pre-analisi della riga a doppia soglia e di smussatura, questo rimane praticamente costante. Grazie all'utilizzo di una finestra di analisi con dimensione finita fissa, il numero di operazioni da calcolare a ogni passaggio rimane pressoché invariato. Di conseguenza, risulta un comportamento lineare come mostrato in Figura 4.19a.

A riprova di ciò, è possibile calcolare il tempo medio in secondi impiegato per la generazione di un punto del grafo, dividendo il tempo totale per il numero di punti. Da questo calcolo, riportato in Figura 4.19b, si desume che, data una superficie, il tempo necessario a generare un singolo punto con l'algoritmo proposto è indipendente dal numero totale di punti da realizzare, e di conseguenza dalla dimensione della maglia impostata in fase iniziale.

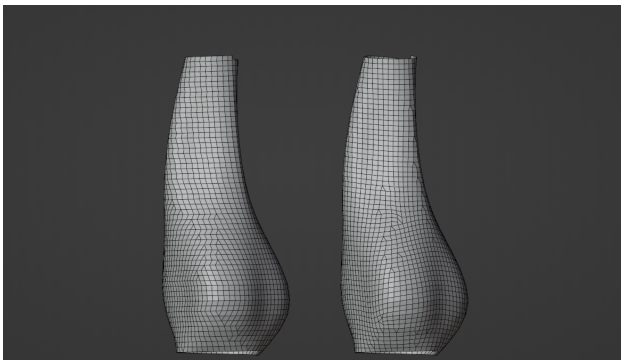
4.2.3 Aspetto visivo

Infine, si valutano visivamente gli oggetti cuciti realizzati seguendo pedissequamente le istruzioni generate, le cui fotografie sono state affiancate alle rispettive versioni realizzate in computer grafica.

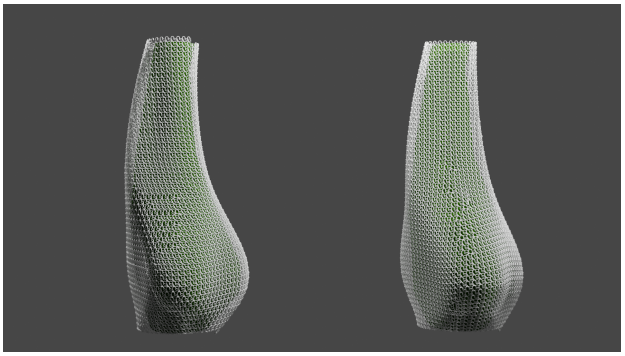
A questo scopo sono state scelte la superficie A, che si sviluppa verticalmente con cambi del raggio asimmetrici, e la superficie C, scelta per mettere in evidenza l'esito in presenza di cambi di direzione.

Per quanto concerne la superficie A mostrata in Figura 4.20, una di quelle estratte dal lavoro di riferimento, si denotano alcune differenze visive, elencate qui di seguito. Innanzitutto, si notano tre gibbosità nella metà bassa a sinistra della superficie, causate dalla congiunzione tra una ampia zona a basso skew e una con un valore più elevato. Secondariamente, nella metà alta della superficie, sulla sinistra dei due oggetti, si denotano curvature opposte: ciò è causato dal fatto che l'algoritmo di riferimento distribuisce le maglie di aumento e diminuzione in corrispondenza di valori di v prossimi a 0.5, disponendo prioritariamente le maglie basse intorno ai valori di v prossimi a 0 o 1.

Riguardo alla superficie C riportata in Figura 4.21, è possibile apprezzare l'utilità delle short row nel mantenimento della curva desiderata. Infatti, l'oggetto realizzato con l'algoritmo di riferimento, a causa delle forze presenti tra le maglie durante il rilassamento del tessuto, approssima un tronco di cono.



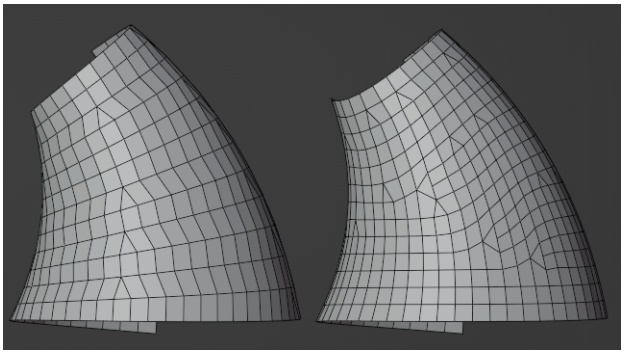
(a)



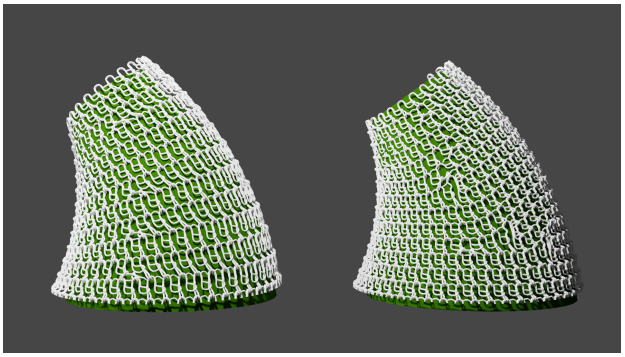
(b)

(c)

Figura 4.20: Confronto dei test eseguiti sulla superficie A: esito dell'algoritmo di riferimento sulla sinistra, esito dell'algoritmo proposto in questa tesi sulla destra; (a) mesh schematica; (b) render di anteprima; (c) oggetto cucito.



(a)



(b)

(c)

Figura 4.21: Confronto dei test eseguiti sulla superficie C: esito dell'algoritmo di riferimento sulla sinistra, esito dell'algoritmo proposto in questa tesi sulla destra; (a) mesh schematica; (b) render di anteprima; (c) oggetto cucito.

Capitolo 5

Conclusioni

L'algoritmo proposto raggiunge lo scopo di integrare le short row in modo incrementale basandosi su una struttura elicoidale. Ad esso è stato accostato un modulo di generazione delle istruzioni testuali con accorgimenti per migliorarne la leggibilità e un modulo di visualizzazione in grado di realizzare una mesh in formato di interscambio `.stl` compatibile con altri programmi di modellazione già esistenti.

L'approccio di calcolo presentato esibisce una accuratezza migliore della lunghezza degli archi di tipo `Wale`, a discapito di una penalizzazione in quelli di tipo `Course`. Ciò, globalmente, si traduce in un risultato con un vantaggio di ridotta entità quando si opera su superfici pressoché cilindriche, con piccole deformazioni; viceversa, quando si prendono in considerazione superfici parametriche più generali, tali che curvino o che il raggio si espanda in modo asimmetrico, si denota miglioramento ingente di questa metrica.

La metrica secondaria utilizzata per valutare la regolarità della mesh, ossia la deformazione di skew, mostra risultati vantaggiosi per superfici pressoché cilindriche ed equivalenti per superfici più complesse.

In virtù dell'uso di un approccio a finestra mobile, i tempi di calcolo ottenuti sono migliorativi rispetto al lavoro di riferimento, scalando in modo lineare rispetto al numero totale di maglie generate.

5.1 Miglioramenti futuri

In conclusione, si espongono di seguito alcuni miglioramenti implementabili in futuro.

Il progetto presentato, grazie a GdNative, è già integrato nel motore Godot Engine come libreria. Ad oggi, il motore è solo usato a scopo di visualizzazione; tuttavia, i metodi della libreria sono esposti e accessibili ad esso, permettendo di creare uno strumento completo e integrato per la prototipazione degli oggetti cuciti. Di conseguenza, un naturale prosieguo è lo studio di una interfaccia per rendere lo strumento software più accessibile agli utenti. In secondo luogo, la tesi esplora l'algoritmo applicandolo a singole superfici parametrizzabili UV; altri ricercatori potranno valutare l'implementazione della gestione delle diramazioni della superficie e del comportamento nei punti di intersezione tra superfici diverse.

In aggiunta alle considerazioni già esplorate sulla rilevanza strutturale dell'uso delle short row e dello studio della deformazione del tessuto, si potrebbero proporre studi per indagare eventuali differenze percettive visive e tattili intercorrenti tra i due algoritmi.

Normalmente, durante la cucitura, è possibile cambiare il filo utilizzato con uno di colore diverso. Ciò potrebbe essere implementato tramite una mappatura di una texture disegnata dall'utente sulla superficie parametrica; l'algoritmo sarà incaricato di suggerire il cambio del colore del filo nelle istruzioni generate. In modo analogo, potrà essere possibile adoperare delle texture integrative per marcare le zone con stili estetici diversi.

Allo scopo di integrare l'output dello strumento sviluppato con l'ecosistema software e hardware già esistente, si potrebbe implementare l'esportazione verso i seguenti formati:

- formato StitchMesh `.smobj`, per quanto concerne la visualizzazione [26];
- formato di istruzioni per telaio automatico con estensione "Knitout" `.k` [27] o "k-code" `.kc` di Kniterate [28];
- formato XML [29].

In ultimo, una delle difficoltà tecniche riscontrate è stata la mancanza del codice sorgente originale dei lavori pubblicati, caratteristica che rallenta il miglioramento di quanto esistente e impone a ogni ricercatore di implementare *ex novo* quanto realizzato da altri per poter comparare gli algoritmi e i rispettivi risultati. Di conseguenza, in ottica di agevolare il lavoro di ricerca degli altri studiosi, il codice sorgente dello strumento software sviluppato in questa tesi verrà pubblicato in un prossimo futuro con licenza libera GPLv3+.

Appendice

Diagrammi UML

I diagrammi riportati sono riassuntivi e non evidenziano in dettaglio tutti i parametri e tipi restituiti di ogni metodo, poiché lo scopo di questa sezione è far trasparire l'organizzazione generale del progetto software, delle interfacce e classi disponibili.

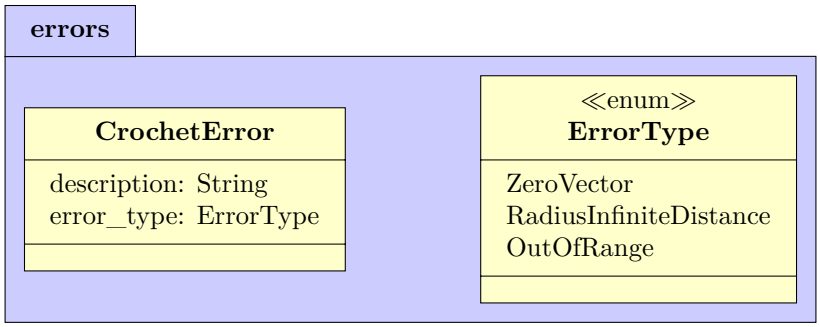


Figura 5.1: Modulo con classe di errore personalizzata

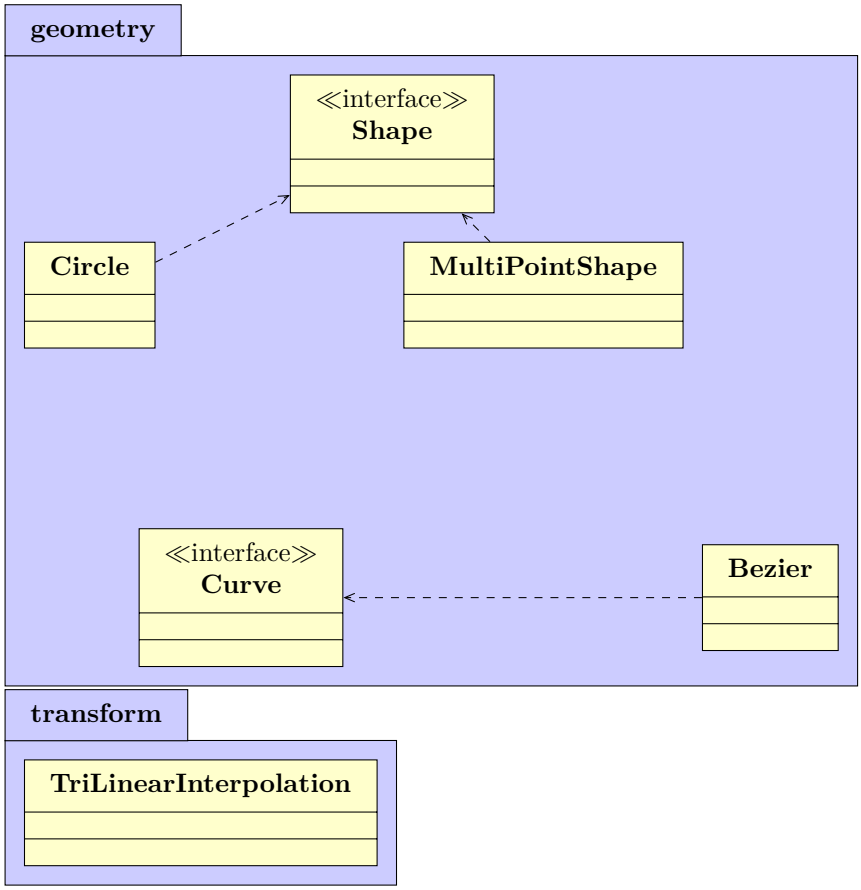


Figura 5.2: Moduli presenti nel file `geometry.rs`

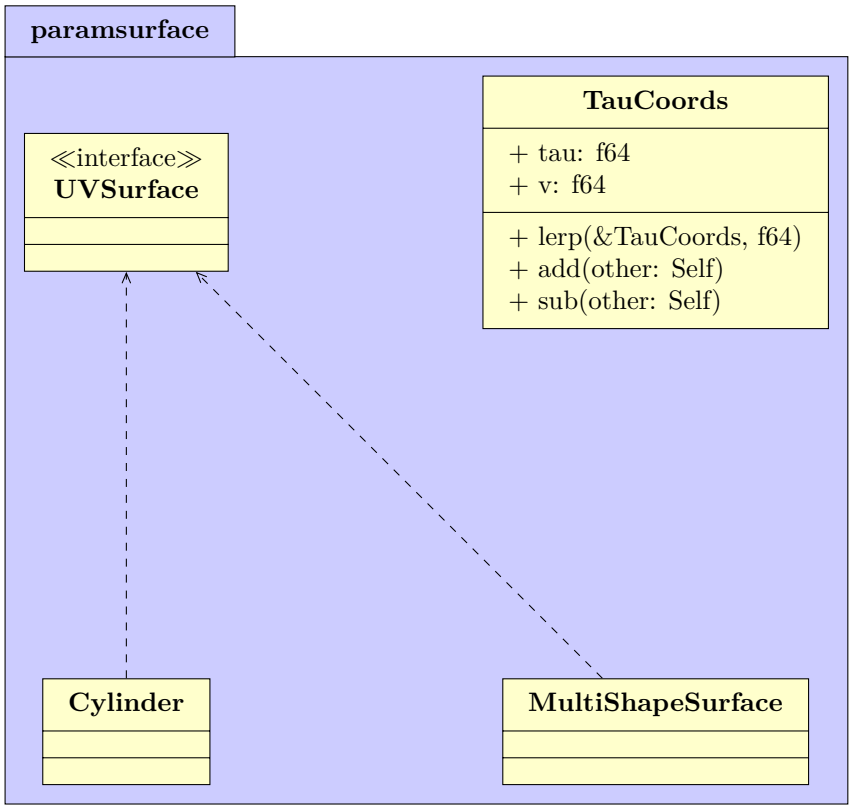
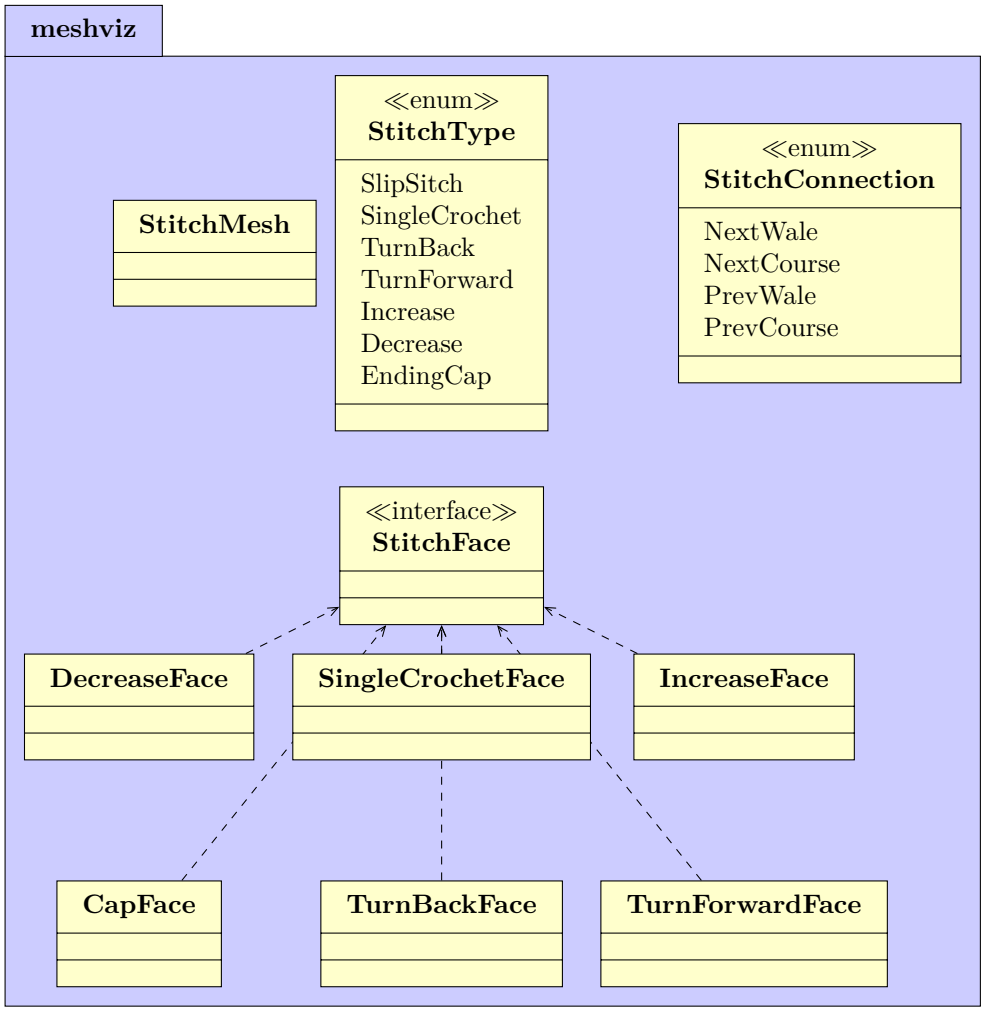
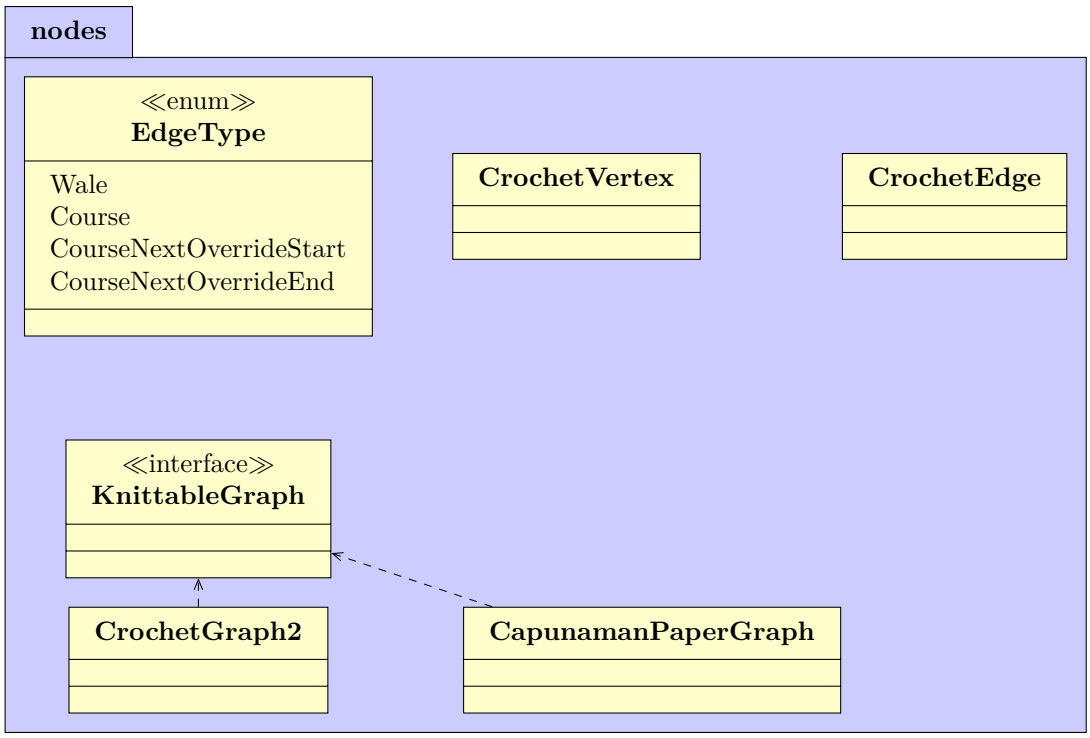
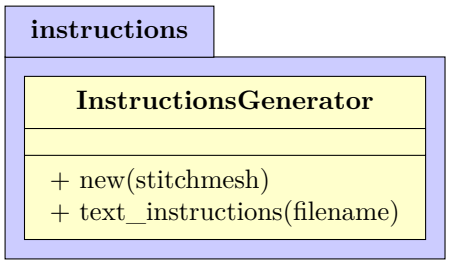


Figura 5.3: Modulo per gestire la superficie


meshviz
≪enum≫
StitchType
SlipSitch
SingleCrochet
TurnBack
TurnForward
Increase
Decrease
EndingCap
≪enum≫
StitchConnection
NextWale
NextCourse
PrevWale
PrevCourse
StitchMesh
≪interface≫
StitchFace
DecreaseFace
SingleCrochetFace
IncreaseFace
CapFace
TurnBackFace
TurnForwardFace


nodes
≪enum≫
EdgeType
Wale
Course
CourseNextOverrideStart
CourseNextOverrideEnd
CrochetVertex
CrochetEdge
≪interface≫
KnittableGraph
CrochetGraph2
CapunamanPaperGraph


instructions
InstructionsGenerator
+ new(stitchmesh)
+ text_instructions(filename)

Bibliografia

[1] Ö. B. Çapunaman, C. K. Bingöl, and B. Gürsoy, "Computing stitches and crocheting geometry," in *International Conference on Computer-Aided Architectural Design Futures*, pp. 289–305, Springer, 2017.

[2] G. Rubio, "Openknit, open source digital knitting." https://openknit.org/.

[3] G. Rubio, "Openknit/bom_wally120.csv at master · g3rard/openknit." https://github.com/g3rard/OpenKnit/blob/master/Wally120/bill_of_materials/BOM_Wally120.csv.

[4] K. team, "Kniterate, the digital knitting machine home." https://www.kniterate.com/.

[5] K. Wu, H. Swan, and C. Yuksel, "Knittable stitch meshes," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 1, pp. 1–13, 2019.

[6] Y. Igarashi, T. Igarashi, and H. Suzuki, "Knitting a 3d model," *Computer graphics forum*, vol. 27, no. 7, pp. 1737–1743, 2008.

[7] M. A. M. Vilela, *Irregular Knitting Pattern generation for segmented triangular meshes.* PhD thesis, University College London, 2019.

[8] Thingiverse, "Digital designs for physical objects." https://www.thingiverse.com/.

[9] "About cc licenses," May 2021.

[10] Y. Igarashi, T. Igarashi, and H. Suzuki, "Knitty: 3d modeling of knitted animals with a production assistant interface.," in *Eurographics (Short Papers)*, pp. 17–20, 2008.

[11] V. Narayanan, L. Albaugh, J. Hodgins, S. Coros, and J. McCann, "Automatic machine knitting of 3d meshes," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 3, pp. 1–15, 2018.

[12] H. M. Osinga and B. Krauskopf, "Crocheting the lorenz manifold," *Mathematical Intelligencer*, vol. 26, no. 4, pp. 25–37, 2004.

[13] M. Popescu, M. Rippmann, T. Van Mele, and P. Block, "Automated generation of knit patterns for non-developable surfaces," in *Humanizing Digital Reality*, pp. 271–284, Springer, 2018.

[14] K. Wu, X. Gao, Z. Ferguson, D. Panozzo, and C. Yuksel, "Stitch meshing," *ACM transactions on graphics*, vol. 37, no. 4, pp. 1–14, 2018.

[15] J. McCann, L. Albaugh, V. Narayanan, A. Grow, W. Matusik, J. Mankoff, and J. Hodgins, "A compiler for 3d machine knitting," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–11, 2016.

[16] S. Greco and P. Valabrega, *Lezioni di geometria*, vol. 2. Libreria editrice universitaria Levrotto e Bella, 1999.

[17] Mozilla Research, "Rust." https://research.mozilla.org/rust/.

[18] The godot-rust developers, "godot-rust." https://godot-rust.github.io/.
[19] A. M. Juan Linietsky and the Godot community, "Arraymesh godot engine 3.2 documentation." https://docs.godotengine.org/en/3.2/classes/class_arraymesh.html#class-arraymesh.
[20] R. Guo, J. Lin, V. Narayanan, and J. McCann, "Representing crochet with stitch meshes," in *Symposium on Computational Fabrication*, SCF '20, (New York, NY, USA), Association for Computing Machinery, 2020.
[21] V. Narayanan, K. Wu, C. Yuksel, and J. McCann, "Visual knitting machine programming," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–13, 2019.
[22] xix xeaon, "Save mesh as stl with godot plugin." https://www.xixxeaon.com/tools/godot-stl-plugin.
[23] engmorph, "Skewness calculation for 2d elements." https://www.engmorph.com/skewness-finite-elemnt.
[24] S. N. L. Cubit documentation, "Metrics for quadrilateral elements." https://cubit.sandia.gov/files/cubit/15.8/help_manual/WebHelp/mesh_generation/mesh_quality_assessment/quadrilateral_metrics.htm.
[25] R. Developers, "Instant in std: : time - rust." https://doc.rust-lang.org/std/time/struct.Instant.html.
[26] Y. I. Vidya Narayanan, ixchow, "Github - textiles-lab/smobj: Documentation and format descriptions for the .smobj (augmented stitch mesh) and .yarns (yarn path) formats." https://github.com/textiles-lab/smobj/.
[27] "The knitout (.k) file format specification." https://textiles-lab.github.io/knitout/knitout.html.
[28] "Knitout-kniterate." https://knit.work/knitout-kniterate/.
[29] E. Zaharieva-Stoyanova and S. Bozov, "Application of xml-based language for digital representation of crochet symbols," *Digital Presentation and Preservation of Cultural and Scientific Heritage. Conference Proceedings. Vol. 7*, 2017.