UNIVERSITÀ DI PISA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA SPECIALISTICA IN INFORMATICA
TESI DI LAUREA

Studio e Implementazione di un Algoritmo per lo Skinning Automatico di Mesh Poligonali Deformabili

Candidato:
Giovanni Sacchetti

Relatori:
Prof. Marcello Carrozzino
Prof. Franco Tecchia
Controrelatore:
Prof. Roberto Grossi

ANNO ACCADEMICO 2013/2014

INDICE

Introduzione

L’animazione tridimensionale è un argomento di grande interesse in svariati ambiti, che vanno dal puro intrattenimento a più serie simulazioni. Esistono numerosissime tecniche di animazione che bilanciano in modo diverso prestazioni, qualità dei risultati, semplicità dell’approccio e possibilità di riutilizzo.

In questo scenario trova uno spazio tutto suo l’animazione di figure umanoidi, i cosiddetti Virtual Humans (VHs). Da circa 50 anni la ricerca ha prodotto una gran quantità di metodi per trattare VHs, adatti alle più diverse esigenze e occupandosi di riprodurre le molteplici sfaccettature del movimento umano. Tra i molti campi di applicazione dell’animazione umana, citiamo alcuni esempi: la simulazione per l’addestramento nell’eseguire operazioni pericolose, difficili o comunque che comporterebbero alti costi; l’analisi dell’interazione umana con oggetti ed ambienti; la creazione di attori virtuali per l’entertainment; lo studio di nuovi metodi di interazione uomo-macchina; le ricostruzioni di attività umane a scopo didattico o per riprodurre e studiare le dinamiche di eventi.

Tra i metodi per modellare il movimento di figure umane, uno dei paradigmi più diffusi si basa sull’utilizzo di due elementi: il primo è un’approssimazione dello scheletro (skeleton) che viene usato per descrivere le animazioni in modo indipendente dalla figura da animare, così da poter essere riutilizzabile con diversi modelli; il secondo è il modello da animare, che viene visto come una superficie detta pelle (skin), la quale si modifica seguendo lo skeleton. Un approccio per far si che la

skin si deformi secondo le ossa (bones) dello skeleton è chiamato skinning. L'algoritmo più diffuso per effettuare lo skinning nel campo delle applicazioni real-time è noto sotto il nome di Linear Blend Skinning (LBS). Tale approccio consiste nell'associare ad ogni vertice della skin un peso che varia tra 0 e 1 per ogni bones, di modo che più il bone ha influenza sul vertice più tale peso è vicino a 1. I pesi vengono poi usati in un blend lineare di trasformazioni rigide per deformare la skin.

Il settaggio dei pesi per ogni vertice avviene generalmente in modo manuale tramite l'ausilio di appositi tool, integrati in strumenti di modellazione. Quest'operazione comporta comunque un certo sforzo da parte del modellatore. Inoltre in certi applicativi si preferirebbe che la determinazione dei pesi avvenisse in modo totalmente automatico e con buoni risultati visivi, senza la necessità di intervento da parte di un modellatore esperto.

In questa tesi si analizzano i metodi più noti per il calcolo automatico dei pesi da utilizzare assieme all'LBS, di modo da ottenere una distribuzione dei pesi il più realistica possibile. Vengono quindi studiati i pregi ed i difetti degli algoritmi esistenti, quindi si propone un nuovo algoritmo per offrire una soluzione ai problemi più importanti emersi dalla precedente analisi.

Struttura della ricerca

- Il primo capitolo dà una breve spiegazione su cosa significhi animare un modello tridimensionale, su cosa sia l'animazione basata su uno skeleton e sullo skinning modellato tramite pesi ed LBS.
- Il secondo capitolo descrive le principali tecniche note per il calcolo automatico dei pesi.
- Nel terzo capitolo verrà descritto uno di questi metodi basato sul concetto di "Atlas of Chart". Verrà data una descrizione di tutte le fasi dell'algoritmo e di

una sua generalizzazione più recente. A questa seguirà una dettagliata analisi delle fasi evidenziando possibili difetti, da cui poi verranno studiate nuove soluzioni per ottenere una versione rielaborata dell'algoritmo.

- Il quarto capitolo si occuperà di descrivere come il metodo rielaborato è stato concretamente implementato in un'applicazione dimostrativa.
- Nel quinto capitolo si analizzeranno i risultati ottenuti comparandoli a quelli di altri metodi.
- Infine verranno tratte le conclusioni complessive del lavoro e si indicheranno possibili sviluppi futuri.

Capitolo 1
L’animazione 3D

1.1 Panoramica sui modelli tridimensionali

La *Computer Graphics* (CG) si occupa di creare immagini ed animazioni con un computer tramite elaborazione di modelli. In particolare per quanto riguarda la CG 3D, questi modelli, sono prevalentemente descritti nello spazio tridimensionale tramite un insieme di triangoli.

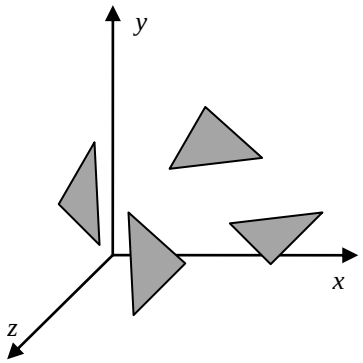


Figura 1.1: *triangoli immersi in uno spazio tridimensionale.*

Tali triangoli vengono anche chiamati in gergo *facce*. Questa rappresentazione è prassi divenuta abituale per diversi motivi sia teorici che pratici. Dal punto di vista teorico possiamo dire che un triangolo è definibile tramite un buon formalismo

essendo un simplesso 2-dimensionale[1] ed inoltre offre meno casi degeneri, per esempio è sempre planare, cioè passa sempre tramite un unico piano. Dal punto di vista pratico un triangolo è rappresentabile tramite una struttura dati semplice e facile da gestire; inoltre l'hardware delle schede grafiche è fortemente orientato alla gestione efficiente di triangoli.

Per ogni faccia vengono memorizzati i 3 vertici (v_1, v_2 e v_3) che la compongono ed ogni vertice è memorizzato tramite le sue coordinate (x,y,z) nello spazio.

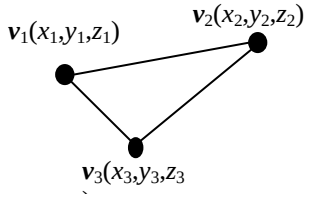


Figura 1.2: *vertici di un triangolo.*

Anche se non sempre, nella stragrande maggioranza dei casi ogni triangolo è posto in modo da avere ogni suo lato (detto *edge*) adiacente con il lato di un altro triangolo.

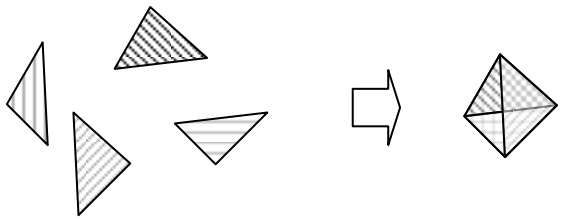


Figura 1.3: *facendo coincidere i bordi dei 4 triangoli si ottiene la rappresentazione di una superficie solida.*

In questo modo si ottengono delle superfici connesse di triangoli. Se di questi visualizziamo solo i bordi delle facce (questo tipo di visualizzazione è detta

[1] Un simplesso è il politopo *n*-dimensionale col minor numero di punti linearmente indipendenti. Ricordiamo che un politopo *d*-dimensionale o *d*-politopo è una generalizzazione della definizione di poligono e di quella di poliedro nello spazio Euclideo reale: i poligoni sono 2-politopi (d = 2), mentre i poliedri sono 3-politopi (d = 3). Un simplesso 2-dimensionale è quindi l'inviluppo di tre punti non allineati, ovvero un triangolo.

wireframe) ci appaiono delle "reti" di triangoli e per questo tali raggruppamenti di triangoli connessi vengono anche chiamati mesh di poligoni, o semplicemente *mesh* (*maglie*, *reti*)[2]. Una mesh triangolare rappresenta quindi un'approssimazione di una superficie reale continua.

Le facce della mesh descrivono come la luce della scena viene riflessa su una superficie (appunto triangolare in questo caso) sino ad arrivare all'occhio dell'osservatore.

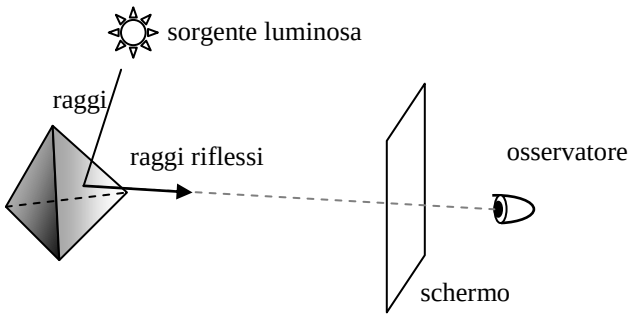


Figura 1.4: *trasformazione del modello in un'immagine.*

La simulazione della riflessione della luce sulla superficie dell'oggetto fino all'occhio dell'osservatore dà così vita ad un'immagine ben definita.

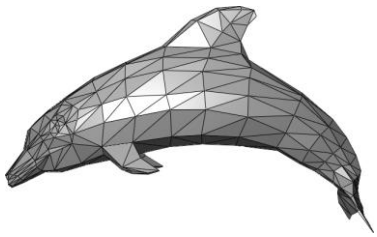


Figura 1.5: *una mesh composta da diversi triangoli connessi che rappresenta un delfino.*

[2] Essendo i triangoli dei simplessi questi raggruppamenti di triangoli sono anche noti come *complessi simpliciali*, ovvero aggregazioni di simplessi che si intersecano tra loro su facce comuni. In un *k*-complesso simpliciale Σ: ogni faccia di un simplesso appartenente a Σ è a sua volta un simplesso appartenente a Σ; l'intersezione tra due simplessi appartenenti a Σ o è vuota o è a sua volta un simplesso appartenente a Σ; ogni simplesso di Σ ha massimo ordine pari a *k*. Una mesh di triangoli è quindi un 3-comlpesso simpliciale ($k = 3$).

1.2 Metodi d'animazione

Appare quindi evidente come le figure che si vogliono rappresentare tramite questi triangoli siano generalmente statiche, poiché le coordinate di ogni vertice sono fisse (in un dato sistema di riferimento). Le tecniche di animazione esistenti consistono nell'alterare in un lasso di tempo spezzettato, o se preferiamo discretizzato, queste coordinate, secondo diversi criteri che dipendono dalla tecnica scelta. Queste alterazioni, o trasformazioni, avvengono dunque per passi discreti: in ogni istante tempo t_1, t_2, t_3, … si applica una trasformazione e si disegna l'immagine. Alternando trasformazione e disegno (detto *rendering*) si dà l'illusione del movimento.

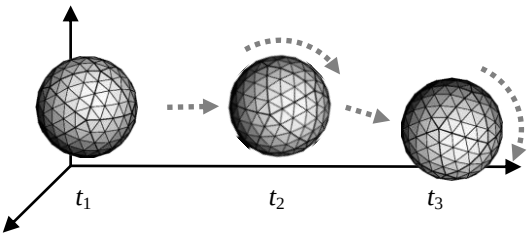


Figura 1.6: *in ogni istante di tempo t_i si applica una o più trasformazioni ai vertici della mesh.*

Esistono diverse tecniche d'animazione che modificano le coordinate dei vertici, tra cui le più famose sono:

- **Interpolazione tra forme**. Consiste nel trasformare le coordinate dei vertici di una forma geometrica nelle coordinate di un'altra forma. Calcolando le coordinate intermedie che ogni vertice deve assumere per passare da una forma all'altra tramite una funzione di interpolazione, si ottengono tutte le trasformazioni necessarie per simulare l'animazione. Tale tecnica viene utilizzata ad esempio molto spesso per l'animazione facciale, ossia per animare mesh (o sottomesh) che rappresentano volti, poiché le espressioni del viso si basano principalmente su contrazioni muscolari e non su spostamenti ossei, mentre è meno adatta per esempio ad animare gli arti.

- **Animazione diretta di vertici o di punti di controllo-spline**. Consiste nello specificare esplicitamente le nuove coordinate che i vertici devono avere nel tempo, oppure tramite punti di controllo che descrivono curve attraverso cui i vertici devono passare.
- **Deformazione basata-sulla-fisica**. Consiste nel trasformare le coordinate dei vertici simulando la reazione ad eventi fisici, quali la pressione, l'urto, ecc.
- **Metodi basati sull'anatomia**. Consiste nel deformare i vertici in modo diverso a seconda del tipo di forma anatomica che rappresenta la mesh o parte di essa. Un esempio può essere la simulazione di effetti muscolari.
- **Metodi misti basati su anatomia e fisica**. Consiste nello sfruttare contemporaneamente le due tecniche sopra descritte per ottenere risultati più realistici.
- **Animazione Scheletrica** (o guidata dalle ossa). Sfrutta la struttura di uno scheletro approssimato per deformare una mesh secondo i suoi arti.

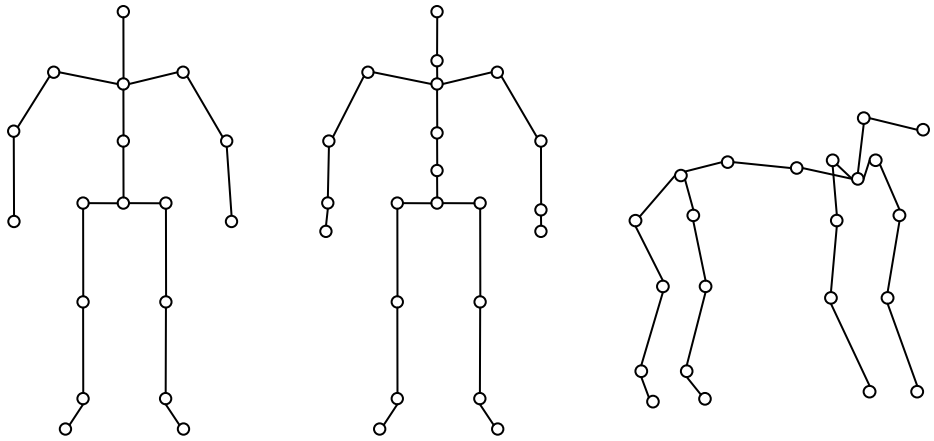
Queste diverse tecniche possono anche essere combinate assieme. Nel nostro caso ci concentriamo sull'ultimo metodo di animazione, l'Animazione Scheletrica, rimandando ad altri studi (Collins, et al., 2001) per una panoramica più ampia dell'animazione di personaggi.

1.3 Animazione Scheletrica

L'Animazione Scheletrica è una tecnica diffusa per deformare le forme articolate. Consiste nell'utilizzo di una struttura scheletrica aggiuntiva detta s*keleton*. Proprio la presenza di questa struttura sottostante rende l'Animazione Scheletrica molto diversa dalle altre forme di animazione.

Da un punto di vista ad alto livello lo skeleton è un'approssimazione di uno scheletro reale. Lo scheletro reale può essere umano, se stiamo trattando modelli umani, ma può essere di qualsiasi genere se trattiamo altre forme anatomiche o anche

solo parti di anatomie, come ad esempio solo la mano. Possiamo avere quindi tanti diversi tipi di skeleton, a seconda dei tipi di anatomie che vogliamo trattare. Inoltre lo skeleton è una forte approssimazione, perché in genere non riprende l'intera forma di uno scheletro reale ma solo la sua struttura e quasi sempre in un numero limitato e semplificato di componenti. In genere questa approssimazione consiste nel rappresentare lo skeleton tramite segmenti, detti *bones* (ossa), aventi in comune i loro vertici estremi, detti *joints* (giunti), che quindi legano i bones in una struttura ad albero. Come ogni struttura ad albero vi sarà quindi un joint radice, ogni joint avrà almeno un joint padre ed potrà avere più joint figli. Alla stessa maniera possiamo definire bones padri e figli (ma non un bone radice). Il numero di bones utilizzabili può variare a seconda di quanti arti mobili vogliamo e sono questi bones che vengono animate. Dunque, oltre a skeleton diversi per anatomie diverse, possiamo avere anche diversi tipi di skeleton per uno stesso tipo di anatomia.



a) skeleton umano b) skeleton umano più complesso c) skeleton animale

Figura 1.7: *alcuni esempi di skeleton diversi: a) umano, b) sempre umano ma di complessità maggiore e c) animale. I segmenti rappresentano i bones, mentre i pallini sono le joints.*

Lo skeleton viene poi utilizzato per simulare i movimenti reali di una certa forma anatomica.

Da un punto di vista implementativo lo skeleton altro non è che una gerarchia di trasformazioni: ogni bone descrive una trasformazione e possiamo pensarlo come la

definizione di un sistema di coordinate a se stante, con una estremità del bone (il *joint padre*) all'origine e la lunghezza del bone distesa lungo un asse del sistema stesso. Ogni bone di trasformazione influenza poi anche i suoi sottobones. Solitamente le trasformazioni nello spazio sono rappresentate da matrici 4 × 4 tramite l'uso di coordinate omogenee[3]. Tali *matrici di trasformazione* convertono le coordinate di un punto, che sono relative al sistema di coordinate locali del bone, in coordinate relative al sistema di coordinate del modello. Vedremo successivamente che vi sono anche altri metodi per rappresentare le trasformazioni oltre a queste matrici. Normalmente un'unica matrice di trasformazione può rappresentare la combinazione di diverse rotazioni, traslazioni e scalature. In generale comunque le trasformazioni cui sono sottoposti i bones di uno skeleton, visti i vincoli di continuità nella catena cinematica, sono per lo più rotazioni rispetto al bone padre: ogni bone figlio rappresenta una rotazione rispetto al sistema di riferimento del bone padre.

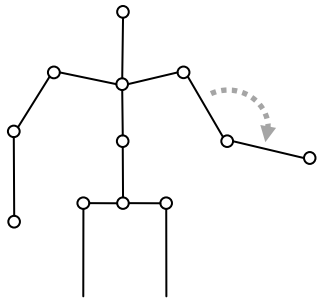


Figura 1.8: *ogni bone figlio rappresenta una rotazione rispetto al bone padre.*

Allo skeleton possono quindi essere legate una serie di animazioni muovendo i suoi bones. I bones possono essere animati direttamente, specificando le nuove

[3] Le coordinate omogenee sono utilizzate per descrivere i punti nella geometria proiettiva. Oggetti rappresentati tramite sequenze $[x_0,\ldots,x_n]$ sono in coordinate omogenee se, quando una sequenza è multipla di un'altra, allora le due sequenze rappresentano lo stesso oggetto, cioè: $\exists\ \lambda \in K$ tale che $[x_{01},\ldots,x_{n1}] = \lambda[x_{02},\ldots,x_{n2}] = [\lambda x_{02},\ldots,\lambda x_{n2}]$. Grazie a questo strumento si possono rappresentare tutte le trasformazioni affini, ovvero tutte le composizioni di una trasformazione lineare con una traslazione, tramite operazioni su matrici. Usando quindi un unico meccanismo si semplifica l'implementazione e si possono ottimizzare le operazioni.

rotazioni bone per bone, oppure indirettamente, tramite ad esempio cinematica inversa, catturando le rotazione attraverso apposite tecniche.

Lo skeleton viene quindi incorporato nel modello e i vertici della mesh vengono legati ad uno o più bones. Dopo tale operazione i vertici seguono ogni trasformazione dei bones a cui sono legati. Quindi ogni rotazione di un bone implica una rotazione anche di tutti i vertici influenzati da quel bone.

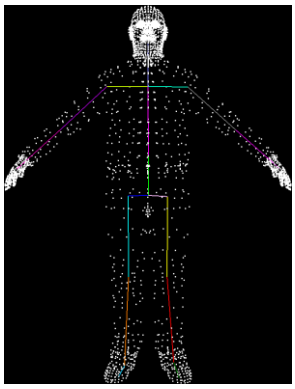


Figura 1.9: *esempio di skeleton immerso in una mesh. Ogni bone ha un colore diverso.*

Osserviamo che la mesh non è animata direttamente, come negli altri metodi di animazione, ma indirettamente tramite il suo legame con lo skeleton, il quale è invece animato direttamente. Uno stesso skeleton può essere legato anche a diverse animazioni. Possiamo poi usare lo stesso skeleton con tutte le sue animazioni per diverse mesh anatomicamente simili. L'importanza dell'animazione scheletrica deriva da questo suo semplice modo di animare mesh aventi una certa struttura anatomica. Ciò rende tale metodo molto adatto ad esempio per l'animazione di personaggi.

Vanno anche considerate le esigenze di renderig: mentre per ambiti dove il rendering avviene off-line (per esempio la produzione di film) sono preferibili tecniche come quelle basate su simulazioni fisiche, qualitativamente migliori ma assai più pesanti, per il rendering on-line è estremamente più adatta l'Animazione Scheletrica. Questo l'ha resa tecnica dominante nel campo dell'animazione real-time, tipica di videogiochi e applicazioni interattive.

1.4 Skinning

In principio mesh e skeleton sono totalmente disconnessi. Dobbiamo quindi specificare come legare mesh e skeleton, ossia come la superficie della mesh si modifica con i movimenti dello skeleton. In altre parole si deve mappare il movimento articolato dello skeleton per ottenere una deformazione della superficie del personaggio. Questo avviene fissando lo skeleton alla superficie in modo che variazioni dello skeleton portino a deformazioni superficiali plausibili ed uniformi ai bones; il movimento dei bones provoca così distorsioni della mesh stessa. Proprio per questo genere di deformazione che ricorda come la pelle si stira a seconda del movimento delle ossa, la superficie viene generalmente chiamata *skin* (pelle) ed il processo per legare ogni vertice della skin allo skeleton è detto *Skinning*[4] o a volte *Enveloping* (*Rivestimento*).

Come è noto il primo articolo (Magnenat-thalmann, et al., 1988) a trattare il problema della deformazione della skin guidata da uno skeleton trattava il caso particolare della mano. Questo documento ha avuto una grande influenza e a partire da esso è stata proposta una grande varietà di metodi. Oggigiorno il metodo più comune per effettuare lo Skinning consiste nello specificare per ogni vertice a "quali" bones è legato e "quanto" vi è legato. Una rotazione è tanto più marcata ed evidente su un vertice quanto più tale vertice è legato al bone che rappresenta tale rotazione. Per esempio un vertice che si trova in prossimità di una giuntura in generale si legherà a tutti i bones incidenti sulla giuntura, in modo che la skin si deformi uniformemente

[4] In realtà, purtroppo, c'è una gran confusione in letteratura nell'uso dei termini skinning e rigging. A volte, specialmente nell'ambito della modellazione, con rigging si intende un unico procedimento che si occupa sia dell'inserimento dello skeleton in una mesh, sia della determinazione dei pesi. Altre volte, in ambito di modellazione, con skinning si intende il processo di calcolo dei pesi, mentre in alcuni articoli di ricerca si intende l'applicazione dei pesi durante l'animazione, come avviene ad esempio nell'LBS. Per quanto ci riguarda in questa ricerca consideriamo il *rigging* come l'inserimento dello skeleton all'interno della mesh, lo *skinning* come l'applicazione dell'algoritmo che deforma la mesh secondo le bones e il *calcolo dei pesi di skinning* come il metodo utilizzato per stabilire i pesi da usare con l'algoritmo di skinning scelto.

attorno alla giuntura. Al contrario un vertice in cima alla testa si legherà esclusivamente al bone della testa.

Per realizzare nel concreto questi legami ogni vertice può essere quindi assegnato a uno o più bone con una serie di *pesi di skinning* (*skinning weights*) che indicano l'influenza che ha ciascuno dei bones su quel vertice. Tali pesi rappresentano quindi la quantità di influenza dei bones sui vertici. I pesi di ogni bone sono indicati con valori che variano tra 0 e 1. Tanto più il vertice è legato ad un bone, tanto più il peso di tale bone per quel vertice è vicino ad 1. Viceversa tanto meno il vertice è influenzato da un bone, tanto più il peso sarà prossimo a 0. Inoltre la somma di tutti i pesi di un dato vertice deve essere uguale a 1.

Ad esempio un vertice posizionato su un ginocchio in prossimità della rotula ed esattamente a metà tra femore e tibia, sarà equamente influenzato dal bone rappresentante il femore e da quello rappresentante la tibia e quindi assegneremo ad entrambi i bones un peso di 0,5 per tale vertice e 0 a tutti gli altri bones.

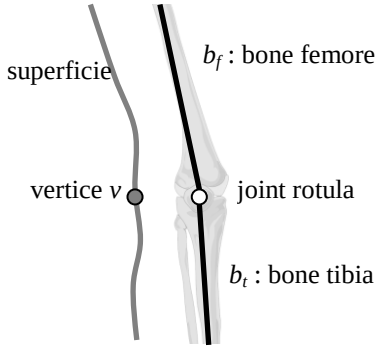


Figura 1.10: *supponendo che il vertice v sulla superficie sia equamente influenzato dai due bones b_f e b_t, i pesi dei due bones per tale vertice saranno: $w_k = w_s = 0{,}5$. Il peso del vertice per un qualsiasi altro bones j sarà $w_j = 0$.*

Notare come abbiamo accuratamente evitato di dire "un vertice equamente distante da femore e tibia". Anticipiamo che tale asserzione infatti andrebbe forse bene in questo particolare esempio, ma non in generale. Approfondiremo il problema nel capitolo successivo relativamente ai difetti della distanza Euclidea.

1.5 Utilizzo dei pesi (LBS)

Una volta stabiliti i pesi si applica un algoritmo che descrive come questi deformano, o per meglio dire trasformano i vertici. L'algoritmo, in assoluto il più diffuso nel campo dell'animazione 3D real-time, è conosciuto sotto diversi nomi, tra cui *Linear Blend Skinning* (LBS), *Skeleton Subspace Deformazione* (SSD), *Single-Weight Enveloping*, *Smooth Skinning*, *Vertex Blending* e a volte (in maniera un po' ambigua) semplicemente *Enveloping*. Tale algoritmo non ha una pubblicazione originale in letteratura, comunque ne esistono buone descrizioni (Praun, et al., 2000) spesso con estensioni e miglioramenti [(Lewis, et al., 2000), (Sloan, et al., 2001), (Wang, et al., 2002), (Kry, et al., 2002), (Mohr, et al., 2003), (Mohr, et al., 2003)].

I vantaggi principali di questo algoritmo sono l'efficienza, la semplicità, il fatto che è già supportato in molti pacchetti di animazione ed infine che è implementabile su GPU. Proprio queste caratteristiche l'hanno reso l'algoritmo più popolare ed utilizzato per tutte le applicazioni real-time, come simulatori o videogame.

Brevemente: l'algoritmo consiste nel determinare la nuova posizione di un vertice combinando linearmente ogni sua trasformazione rigida dei bones che lo coinvolgono. Ad ogni vertice viene cioè associato un blend lineare pesato delle trasformazioni di ogni bone coinvolto. L'algoritmo parte da una *posa di riposo* o *bind pose* ("posa di associazione", cosi detta perché anche usata per legare i vertici alle bones in una fase precedente all'applicazione dell'LBS) della mesh e dello skeleton. Per generare una nuova deformazione dalla bond pose, per ogni vertice, viene calcolata di volta in volta e separatamente una trasformazione diversa per ogni bone. Si prendono quindi tutte queste trasformazioni rigide e si applicano al vertice, pesandole diversamente a seconda di quanto ogni bone influenza realmente il vertice.

Nella posa di riposo ogni vertice ha posizione $\hat{v}$ e ogni bone *j* ha trasformazione $\hat{G}_j$. Si determina quindi quale è la trasformazione del vertice separatamente per ogni

bone *j*, applicando la trasformazione inversa del bone *j* alla posizione del vertice nella posa di riposo $\hat{v}$:

$$\hat{v}_j = \hat{G}_j^{-1}\hat{v}$$

Per ottenere una nuova posizione del vertice si applica la nuova trasformazione G_j:

$$v_j = G_j\hat{v}_j = G_j\hat{G}_j^{-1}\hat{v}$$

Quindi v_j è la posizione del vertice quando viene spostato rigidamente con il bone *j*. Avremo quindi tanti vertici v_j quanti sono i bones e tali vertici saranno le trasformazioni rigide di $\hat{v}$: una per ogni bone appunto.

Ora bisogna decidere come usare queste trasformazioni per ogni vertice. Un caso estremo è quello in cui decidiamo che ogni vertice è legato ad un solo bone *k*, o meglio ad una sola matrice di trasformazione. Avremo così una sola trasformazione $(G_k\hat{G}_k^{-1})$ e quindi un solo v_k che coinciderà da solo con la nuova posizione v del vertice.

$$v = G_k\hat{G}_k^{-1}\hat{v} = v_k$$

Questo significa che con questa scelta estrema i vertici seguono rigidamente un solo bone e non vi è una deformazione della superficie.

Se invece si utilizza l'LBS per ottenere la posizione del vertice rispetto a più bones che lo influenzano, si usano tutte le posizioni rigide v_j combinate con i pesi di ogni rispettivo bone. Si ottiene quindi l'equazione che definisce l'LBS e che viene applicata indipendentemente ad ogni v:

$$v = \sum_{j=0}^{b-1} w_j G_j \hat{G}_j^{-1} \hat{v}$$

Ponendo $C_j = G_j \hat{G}_j^{-1}$ per avere una sola trasformazione, possiamo anche scrivere:

$$v = \sum_{j=0}^{b-1} w_j C_j \hat{v}$$

I pesi devono essere valori normalizzati: $\sum_{j=0}^{b-1} w_j = 1$. Notiamo inoltre che tale equazione viene applicata indipendentemente ad ogni vertice v, ovvero per calcolare la nuova posizione del vertice v non serve conoscere nulla di tutti gli altri vertici. Il caso delle trasformazioni rigide dei vertici è in effetti un caso particolare dell'LBS in cui per ogni vertice vi è un solo bone k per cui vale $w_k = 1$ e per tutte le altre bone i pesi sono uguali a 0:

$$v = \sum_{j=0}^{b-1} w_j G_j \hat{G}_j^{-1} \hat{v} = \left(\sum_{j=0, j \neq k}^{b-1} 0 \cdot G_j \hat{G}_j^{-1} \hat{v} \right) + 1 \cdot G_k \hat{G}_k^{-1} \hat{v} = G_k \hat{G}_k^{-1} \hat{v} = v_k$$

Riassumendo, in modo informale possiamo dire che per generare ogni movimento si parte sempre dalla posa di riposo ($\hat{v}$) e si calcola la nuova posizione usando diverse trasformazioni (C_j) combinandole secondo un peso che gli è stato assegnato (w_j).

È possibile impostare un numero massimo di bones che influenzano il vertice. Un numero basso di bones produce deformazioni rigide e marcate; in tal caso molti pesi si annullano e quindi i pesi vanno ribilanciati portando un incremento dei pesi relativi ai bones coinvolti. Il vantaggio è che limitando il numero di bones coinvolti a 2 o 4 l'LBS si può implementare su GPU (programmando vertex shader tramite CG-*language* ad esempio, un linguaggio high-level orientato allo shading che permette di programmare ed effettuare calcoli sulla GPU) parallelizzando e velocizzando così i calcoli. Viceversa un numero elevato di bones utilizzate produce deformazioni

smussate che possono anche essere poco realistiche, oltre ad accentuare in certi casi problemi che vengono descritti di seguito.

1.5.1 Problemi dell'LBS

Nonostante la sua diffusione l'LBS ha numerose lacune. Per esempio tra i difetti secondari notiamo che ha un limitato potere di modellazione e non può riprodurre in modo attendibile certi effetti, come il rigonfiamento muscolare. Tuttavia tra i vari difetti i più problematici sono due: il *collapsing elbow* e il *candy-wrapper*. Entrambi questi artefatti causano delle *perdite di volume*, ovvero il volume attorno al joint si riduce in modo innaturale. Il collapsing elbow (gomito collassante) avviene quando un arto viene piegato molto.

Figura 1.11: *perdita di volume dovuta ad un piegamento eccessivo (collapsing elbow).*

Il candy-wrapper (incarto di caramella) avviene quando un arto si torce.

Figura 1.12: *perdita di volume dovuta ad una torsione (canddy-wrapper).*

Entrambi sono causati dal fatto che l'LBS sfrutta una combinazione lineare di matrici di rotazione che, quando si usa un grande angolo di rotazione tra loro, comportano inevitabili perdite di volume. Questo perché l'interpolazione lineare di matrici è diversa dall'interpolazione lineare delle loro rotazioni e così le articolazioni tendono a collassare.

Prendiamo per esempio un vertice influenzato da due soli bones con il medesimo peso, quindi con $w_1 = w_2 = 0{,}5$. Supponiamo di effettuare rotazioni solo attorno all'asse *y*: una matrice di trasformazione per tale rotazione ha forma:

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dove θ è l'angolo di rotazione applicato. Teniamo fisso uno dei 2 bones, per esempio il primo ($j = 1$). Dunque la rotazione applicata da tale bone è di 0 gradi, mentre ruotiamo l'altro bone ad esempio di 90 gradi. L'interpolazione lineare delle matrici sarà:

$$(w_1 C_1 + w_2 C_2) = 0{,}5 \begin{bmatrix} \cos 0 & 0 & -\sin 0 & 0 \\ 0 & 1 & 0 & 0 \\ \sin 0 & 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + 0{,}5 \begin{bmatrix} \cos 90 & 0 & -\sin 90 & 0 \\ 0 & 1 & 0 & 0 \\ \sin 90 & 0 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0{,}5 & 0 & 0 & 0 \\ 0 & 0{,}5 & 0 & 0 \\ 0 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0 & 0{,}5 \end{bmatrix} \begin{bmatrix} 0 & 0 & -0{,}5 & 0 \\ 0 & 1 & 0 & 0 \\ 0{,}5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}5 \end{bmatrix}$$

$$= \begin{bmatrix} 0{,}5 & 0 & -0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0{,}5 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Notiamo come la matrice di trasformazione del primo bone non sia altro che l'identità ($C_1 = I$).

Interpolando invece la rotazione della prima matrice che è di 0° e la rotazione della seconda matrice che è di 90°, si avrebbe una rotazione di 45° (sempre attorno all'asse *y*) che è definita dalla seguente matrice di rotazione:

$$\begin{bmatrix} \cos 45 & 0 & -\sin 45 & 0 \\ 0 & 1 & 0 & 0 \\ \sin 45 & 0 & \cos 45 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tale matrice è ovviamente diversa da quella ottenuta precedentemente tramite l'interpolazione delle matrici sfruttata dall'LBS.

Dunque interpolando linearmente le matrici si ottiene una matrice diversa da quella che si otterrebbe interpolando le rotazioni. Un'ulteriore analisi delle perdite di volume dovute all'uso di matrici di rotazione nell'LBS è descritta nella prima parte dell'APPENDICE A.

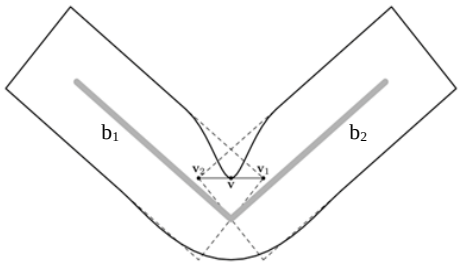


Figura 1.13: v_1 *indica la posizione del vertice secondo una trasformazione rigida rispetto al bone* b_1*, mentre* v_2 *indica la posizione del vertice secondo una trasformazione rigida rispetto al bone* b_2*. La combinazione di queste due trasformazioni rigide da la nuova posizione* v *e genera una perdita di volutme.*

Ciò significa che l'LBS funziona abbastanza bene per angoli di rotazione minori, mentre dà problemi per angoli grandi di rotazione. È da notare anche che maggiore è il numero di bones che coinvolgono un vertice con un peso non irrilevante, maggiore sarà anche il collasso quando queste ruotano tutte assieme (vedremo come questo è spesso evidente in zone come l'inguine: zona normalmente influenzata dai bones delle cosce e del bacino, i quali solitamente ruotano molto tra loro).

Come abbiamo detto esistono numerosi varianti dell'LBS che mirano a risolvere o quantomeno a ridurre questi difetti. Di seguito ne citiamo alcuni con una breve descrizione:

- **Innesto di muscoli**. (Wang, et al., 2010)
  Si inseriscono meccanismi per simulare il gonfiamento muscolare. Questo è un ibrido tra i metodi basati su anatomia e l'LBS standard. Ha il vantaggio di essere implementabile su GPU e la qualità aumenta, tuttavia aumentano anche i costi.
- **Uso di reticoli** (Chen, et al., 2011).
  Tramite un reticolo di supporto formato da *voxel*[5] si può velocizzare lo skinning, applicare deformazioni secondarie e preservare lo spessore del volume. Il problema cardine di questo metodo è la scelta della granularità dei voxel.
- **Interpolazioni non-lineari**.
  Come spiegato i problemi dell'LBS sorgono poiché l'interpolazione lineare di matrici è diversa dall'interpolazione delle loro rotazioni. Con questi metodi si sfruttano quindi interpolazioni non-lineari per superare i difetti. Questa categoria di metodi è comunque meno efficiente dell'LBS standard.
  - **Bone aggiuntivi** o **interpolazione sferica** (Mohr, et al., 2003).
    Si introducono pseudo-bones che vengono ruotati di metà dell'angolo di rotazione della giuntura. Questa è un'*interpolazione sferica*, non lineare e permette di evitare artefatti di collapsing elbow. Per di più si possono introdurre pseudo-bones per avere anche scalature piuttosto che rotazioni, così da supportare effetti di rigonfiamento muscolare. Il vantaggio di questo metodo è che viene alterato solo il modello dello skeleton, e non

[5] . La *voxellizzazione* di un volume consiste nell'approssimare il volume stesso tramite dei cubi di dimensioni identiche; tali cubi sono appunto chiamati *voxel* (contrazione di *volumetric pixel* o *volumetric picture element*, per la loro corrispondenza con i pixel ma in uno spazio 3D).

l'LSB. Tuttavia, il numero di influenze ossee per vertice aumenta, il che riduce le prestazioni di rendering e può anche rendere impraticabile l'accelerazione hardware (poiché, nella programmazione della GPU, si può passare solo un numero fisso di attributi per vertice).

- **Operatore di fusione di matrici** (Magnenat-Thalmann, et al., 2004).
  Utilizzando un operatore di fusione tra matrici come quello descritto in "Linear combination of transformations" (Alexa, 2002) per eseguire l'interpolazione. In tal modo si migliorano i risultati a discapito dei costi.
- **Spherical Blend Skinning** o **quaternioni duali** (Zhai, et al., 2011).
  Si calcola un'interpolazione lineare di quaternioni anziché di matrici (si veda APPENDICE A). I risultati sono meno precisi dell'interpolazione sferica e si trattano solo rotazioni, ma il costo computazionale è più basso.

- **Pesi aggiuntivi**.
  Sono metodi che aumentano il numero di pesi utilizzati nell'equazione dell'LBS.
  - **MWE: MultiWeight Enveloping** (Wang, et al., 2002).
    I pesi scalari vengono sostituiti con matrici di peso 4×4. I pesi aggiuntivi producono effetti non-lineari; si hanno così gradi di libertà aggiuntivi che evitano molti dei difetti dell'LSB. Di contro il metodo necessita di più memoria, più banda e tempi maggiori.
  - **AS: Animation Space** (Merry, et al., 2003).
    Ogni peso è sostituito da un vettore di 4 componenti, similmente all'MWE ma con un numero di pesi tre volte inferiore. Anche se il numero di pesi aumenta comunque di 4 volte non è più necessario memorizzare la posa di riposo. Allo stesso modo l'overhead dovuto ai parametri in più si bilancia col fatto che il metodo necessita di una moltiplicazione scalare in meno dell'LBS standard.

Alcune di queste tecniche possono essere anche combinate tra loro per bilanciare qualità ed efficienza. Tra tutte abbiamo deciso di rifarci alla famiglia di metodi che

sfruttano interpolazioni non-lineari, in particolare ai metodi che utilizzano quaternioni per descrivere rotazioni. Tali metodi bilanciano bene qualità ed efficienza. Questa scelta è stata fatta anche per la disponibilità di codice già pronto, orientato all'uso di quaternioni per animazioni scheletriche. Un approfondimento dell'uso di quaternioni si può trovare in APPENDICE A.

1.6 Obbiettivo: calcolo automatico dei pesi

Il lavoro svolto in questa tesi rientra in un progetto più ampio. Il progetto complessivo mira a creare un'estensione del framework XVR (eXtreme Virtual Reality) (Carrozzino, et al.). Questo è un ambiente di sviluppo integrato per l'implementazione veloce di applicazioni per ambienti virtuali sviluppato da VRMedia. Usando un'architettura modulare e un linguaggio di scripting proprio, attualmente, XVR è una componente ActiveX che può essere incorporata in molte applicazioni, come ad esempio il browser web Internet Explorer. ActiveX è infatti una tecnologia che permette l'estensione di potenzialità di applicazioni, come comandi, funzionalità e caratteristiche. Il linguaggio di programmazione di XVR è simile al C++ e possiede vari costrutti e comandi appositamente creati per applicazioni di realtà virtuale, come animazioni tridimensionali, effetti sonori posizionali e interazioni dell'utente. Inoltre è possibile espandere XVR attraverso moduli aggiuntivi che ne aumentano le funzionalità. Un esempio è il modulo HALCA che consente di visualizzare e gestire avatar conformi al formato CAL3D.

L'estensione che puntiamo a creare per XVR permetterà di generare in modo automatico avatar tridimensionali possibilmente simili agli utenti, per poter poi utilizzare tali avatar in una o più applicazioni di simulazione grazie appunto al modulo HALCA. Per fare ciò, come prima cosa, la nostra estensione dovrà acquisire tramite un dispositivo di scansione tridimensionale una mesh e calcolarne opportunamente il relativo skeleton dell'utente, quindi dovrà legare lo skeleton alla mesh calcolando in

maniera automatica i pesi, ed infine caricherà l'avatar nell'applicazione, che poi verrà usato insieme ad un sistema di animazione (o tramite animazioni predefinite o tramite l'acquisizione real-time dei movimenti dell'utente con uno strumento di motion capture). In particolare ciò che ci prefiggiamo in questa ricerca è quindi trovare un metodo per legare la mesh allo skeleton, ossia un metodo possibilmente automatico di calcolo dei pesi, che riesca a fornire risultati soddisfacenti.

Nell'ambito di questa tesi daremo per già acquisiti e presenti sia la mesh e che il relativo skeleton, attraverso operazioni che avverranno in una prima fase nella quale verrà chiesto agli utenti di mettersi in una posa specifica per l'acquisizione della loro immagine, ad esempio in T-pose (cioè dritti, in piedi e con le braccia allargate orizzontalmente in modo da formare appunto una T con il corpo), oppure in A-pose (con gambe leggermente divaricate e braccia allargate), o in una posa ibrido tra la A-pose e la T-pose, o comunque in una posa dove gli arti siano ben separati e distinguibili.

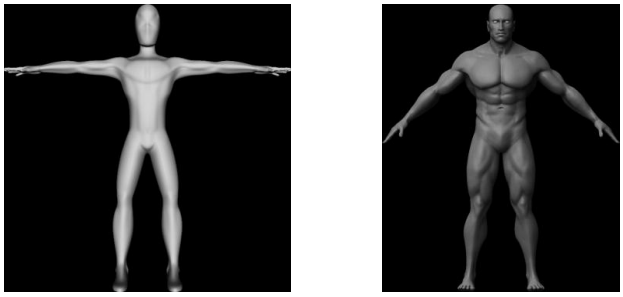


Figura 1.14:*la figura a sinistra è in T-pose, mentre quella a destra è in A-pose.*

Inoltre si suppone che gli utenti non indossino né tengano oggetti troppo complessi, così da garantire una buona acquisizione della mesh. Date queste premesse possiamo fare delle assunzioni sulle mesh che ci semplificano il problema: mesh e skeleton sono in una posa standard, sono antropomorfi e lo skeleton è già perfettamente inserito all'interno della mesh. Trattiamo quindi mesh con relativi skeleton di esseri umani in una posizione standard.

Capitolo 2
Tecniche di calcolo automatico dei pesi di Skinning.

Il nostro lavoro è focalizzato sul calcolo automatico dei pesi per lo skinning. Ovvero trattiamo le tecniche più conosciute per calcolare per ogni vertice i pesi di ogni bone, in maniera automatica o semi-automatica.

Trovare questi pesi non è semplice. Com'è ovvio l'assegnazione manuale dei pesi risulta eccessivamente onerosa per un qualsiasi modellatore, basti pensare a mesh composte da milioni di vertici. Tutt'oggi, anche quando aiutati da un supporto automatico, gli animatori spesso sono costretti a ritoccare a mano i valori dei pesi calcolati, il che può essere comunque un'operazione noiosa e che porta via molto tempo ed energie. Nel nostro particolare caso poi cerchiamo un metodo che dia risultati soddisfacenti in maniera più automatica possibile.

Ci sono diverse proprietà che desideriamo per i pesi. In primo luogo vorremmo un'assegnazione geometricamente coerente, per esempio ci aspettiamo che un vertice sulla fronte sia prevalentemente legato al bone corrispondente alla testa e sia totalmente slegato da altri bones come quelli dei piedi, delle gambe o delle mani. Un'altra proprietà che vogliamo è che i valori dei pesi non devono dipendere dalla risoluzione della mesh. Inoltre, per ottenere buoni risultati visivi, i pesi devono variare dolcemente lungo la superficie. Infine, per evitare artefatti di piegatura (collapsing elbow), la larghezza di una transizione tra due bones che si incontrano in un joint dovrebbe essere approssimativamente proporzionale alla distanza tra il joint e la superficie: in questo modo infatti come vedremo tali artefatti, inevitabili per la natura intrinseca dell'LBS standard, vengono in un qualche modo mascherati.

Nella nostra ricerca abbiamo individuato principalmente 4 metodi basati su:

1. **Distanza Euclidea**.
2. **Distanza Geodetica**.
3. **Diffusione del Calore**.
4. **Addestramento basato-su-esempi**

Ognuno di questi metodi si limita a calcolare i pesi per lo skinning ed è quindi integrabile con qualsiasi estensione dell'LBS citata sopra. Per evitare confusione, esprimeremo nelle formule l'indice di un vertice tramite *i* e l'indice di un bone tramite *j*, anche quando magari nei documenti originali l'uso degli indici è invertito o comunque diverso.

2.1 Distanza Euclidea

Parlando di distanza Euclidea in realtà più che di un metodo parliamo di una famiglia di metodi. Ci sono diversi modi infatti di utilizzare la distanza Euclidea per calcolare i pesi. Questi metodi risultano in assoluto i più semplici e veloci rispetto alle

altre alternative, sia in termini di realizzazione che in termini di costo. Essi si fondano sul principio secondo cui "un vertice è maggiormente influenzato dai bones ad esso più vicini". Per questo tali metodi sfruttano la distanza Euclidea per stabilire la vicinanza di un bone al vertice. Il peso del bone sarà inversamente proporzionale alla distanza, dopo di che si normalizzano i pesi rendendo la loro somma pari ad 1.

2.1.1 Distanza Euclidea semplice

Come abbiamo detto esistono diverse formule che sfruttano la distanza Euclidea. Di seguito indichiamo con w_j il peso di un bone *j* su un vertice fissato ed indichiamo con $d(j)$ la distanza Euclidea tra il vertice e il bone *j*.

Una formula basata sulla distanza Euclidea dai bones è la seguente (Wang, et al.):

$$w_j = \frac{\sum_{k=0}^{b-1} d(k) - d(j)}{(b-1)\sum_{k=0}^{b-1} d(k)}$$

In questo caso w_j è calcolato come un rapporto: al denominatore abbiamo la somma di tutte le distanze da tutti i bones coinvolti moltiplicato il numero di bones coinvolti, mentre al numeratore abbiamo la somma di tutte le distanze da tutti i bones escluso il bone corrente *j*.

Altri approcci circondano ogni bones con *involucri* detti *capsule interne* e *capsule esterne* e sfruttano la distanza Euclidea in modo diverso a seconda della posizione dei vertici rispetto a queste capsule (Komura). Se il vertice è dentro solamente ad un singola capsula interna, il peso per il corrispondente bone è settato a 1, mentre per tutti gli altri bones è 0. Se il vertice si trova dentro più capsule interne, viene calcolata la distanza da ogni bone e i pesi vengono poi impostati con un valore inversamente proporzionale alla distanza (poiché una distanza maggiore indica minor peso) tramite normalizzazione: $w_j = \sqrt{\frac{\frac{1}{d_j}}{\sum_{k=0}^{b-1}\frac{1}{d_k}}}$.

Infine se il vertice è dentro la capsula interna di un solo bone di indice *s* e nelle capsule esterne di altri bones, si utilizza nuovamente la distanza, ma con una funzione di caduta di forza $f(x)$ (*fall-off function*):

$$w_s = \sqrt{\frac{\frac{1}{d_s}}{\left(\sum_{k\neq s,k=0}^{b-1} f(d_k)\cdot\frac{1}{d_k}\right)+\frac{1}{d_s}}} \quad \text{e} \quad w_j = \sqrt{\frac{f(d_j)\cdot\frac{1}{d_j}}{\left(\sum_{k\neq s,k=0}^{b-1} f(d_k)\cdot\frac{1}{d_k}\right)+\frac{1}{d_s}}} \quad \text{con } j \neq s$$

$$\textit{fall-off function:} \qquad f(d_j) = \begin{cases} 1 & \text{se } d_j < r_s \\ 0 & \text{se } d_j > r_j \\ 1 - \frac{d_j - r_s}{r_j - r_s} & altrimenti \end{cases}$$

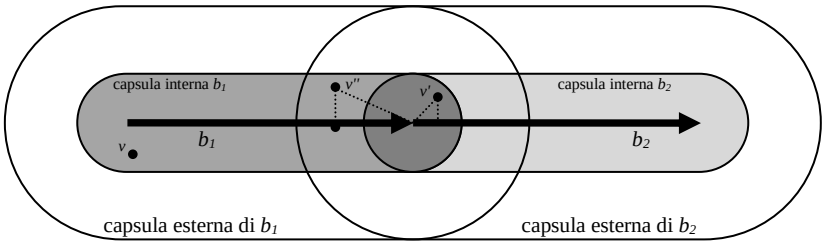


Figura 2.1: *il vertice v è dentro ad un'unica capsula interna, quella di* b_1*, quindi avremo* $w_1 = 1$ *e* $w_2 = 0$*; il vertice v' è dentro a più capsule interne, quindi verranno usate le semplici distanze Euclidee ottenendo* $w_1 = \sqrt{\frac{1/d_1}{1/d_1+1/d_2}}$ *e* $w_2 = \sqrt{\frac{1/d_2}{1/d_1+1/d_2}}$*; il vertice v'' è dentro alla capsula interna di* b_1 *e dentro la capsula esterna di* b_2*, quindi verrà usata la funzione di fall-off ed i pesi saranno* $w_1 = \sqrt{\frac{1/d_1}{1/d_1+f(d_2)/d_2}}$ *e* $w_2 = \sqrt{\frac{f(d_2)/d_2}{1/d_1+f(d_2)/d_2}}$.

Vi sono anche metodi che tengono in considerazione la lunghezza dei bones, per esempio (Dominio, 2010):

$$w_{ij} = e^{-\frac{d(i,j)^2}{2\sigma^2}}$$

dove $d(i,j) \in$ IR è la distanza (norma Euclidea) del vertice *i* dal bone *j* e $\sigma \in$ IR è un valore proporzionale alla lunghezza del bone.

Poco più avanti, parlando di euristiche, vedremo ancora un'altra formula parametrizzabile (su un valore α). Inoltre, nel terzo capitolo, vedremo che nella

generalizzazione del metodo basato su Atlante di Carte il calcolo dei pesi può essere applicato con qualsiasi distanza, compresa l'Euclidea (ereditandone pregi e i difetti).

2.1.2 Problematiche della distanza Euclidea

Nonostante la distanza Euclidea sia usata molto spesso come metodo per calcolare i pesi per la sua estrema elementarità, essa presenta diversi difetti. Questi difetti derivano dal fatto che la distanza Euclidea non tiene conto della geometria o, se preferiamo, dell'anatomia del soggetto. Infatti l'asserzione di fondo non sempre è verificata: non è detto che un vertice debba essere maggiormente influenzato dai bones ad esso più vicini.

Il caso più evidente in cui questo approccio fallisce capita quando si da influenza maggiore ad un bone a cui il vertice non appartiene anatomicamente. Si pensi ad esempio ad un personaggio con una pancia grossa ed un braccio più sottile steso lungo il fianco: in questo modello se prendiamo un vertice nella parte più esterna della pancia, sul fianco, questo risulterà più vicino al bone del braccio che al bone dell'addome. Così, nonostante il vertice appartenga al fianco avrà peso maggiore il bone del braccio e non un bone del fianco.

Figura 2.2: *i vertici sul fianco sono più vicini ai bones sul braccio che ai bones del fianco stesso. Questo genera un assegnamento dei pesi anatomicamente scorretto.*

Per lo stesso motivo le parti di superficie dove si ripercuote maggiormente la flessione dell'arto possono non corrispondere esattamente alle zone che ci aspetteremmo. In pratica si individuano *aree di massima flessione* poco realistiche.

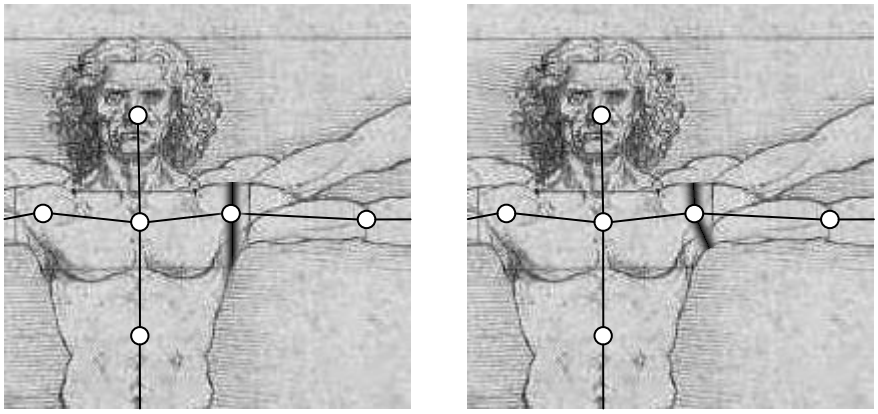


Figura 2.3: *nella figura destra la fascia nera contrassegna la parte di superficie di maggiore flessione individuata tramite la distanza Euclidea. Tale area non è la più anatomicamente adatta come superficie di maggior flessione, come potrebbe essere invece quella indicata nella figura a destra.*

Questo non è un grosso errore dal punto di vista matematico, né per la morbidezza del risultato, ma è poco realistico dal punto di vista anatomico. Per esempio nel caso il modello abbia un torace molto più ampio rispetto alle braccia (cosa che si verifica naturalmente nell'anatomia umana), muovendo un braccio si potrebbe generare una flessione della superficie non esattamente corrispondente alla zona dell'ascella, ma leggermente spostata da questa.

Figura 2.4: *la distanza Euclidea individua una zona di massima flessione non precisamente sull'ascella, ma più spostata: questo comporta una deformazione non realistica per l'anatomia quando l'arto viene mosso.*

Anche quando si riesce a far si che un vertice venga pesato maggiormente dal suo bone principale si possono creare problemi. Ad esempio se il modello è in T-pose le

gambe sono molto vicine tra loro e dunque, anche se la distanza Euclidea per ogni vertice ci garantisce (in questo caso giustamente) che il bone più vicino è quello più influente, i bones di una gamba finiranno per influenzare molto i vertici dell'altra ed anche se tale influenza è minore rispetto a quella del bone principale in realtà dovrebbe essere totalmente nulla, dato che gli arti sono completamente separati.

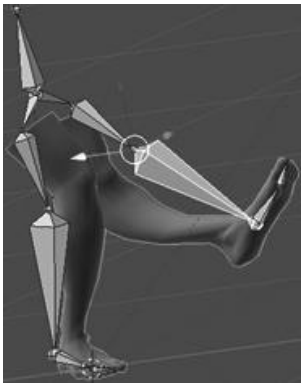


Figura 2.5: *i bones della gamba destra influenzano troppo i vertici della gamba sinistra a causa della loro vicinanza nonostante le due gambe siano totalmente separate e quindi la gamba si deforma in modo irrealistico.*

Tutte queste anomalie in realtà sono manifestazioni dello stesso unico difetto: nel calcolare la distanza Euclidea, si valuta solamente il singolo vertice e i bones, ma non si tiene conto in alcun modo della geometria e dell'anatomia del modello. Esiste un video (Roselle, 2012) che mostra questi difetti in un'implementazione per Maya, oltre a come questi vengono risolti dal metodo della diffusione del calore che descriveremo in seguito.

D'altro canto non considerare la geometria può portare anche alcuni vantaggi sotto altri aspetti: le mesh non devono per forza essere *manifold*[6], chiuse e completamente connesse per garantire la navigabilità (elementi che, come vedremo,

[6] Una superficie manifold è una superficie in cui ogni punto ha un intorno locale topologicamente ad un disco unitario. In oggetti manifold dunque gli edge possono appartenere al massimo solo a due facce. L'utilizzo di mesh manifold comporta diversi vantaggi. Ad esempio un contorno manifold separa in modo non ambiguo una regione interna da una esterna. Inoltre consentono l'utilizzo di rappresentazioni della mesh efficienti per la sua navigazione topologica.

sono invece spesso necessari in altri metodi). Addirittura questi metodi possono funzionare anche su mesh dove ogni faccia è una tripla di vertici a-sé-stante (una nuvola di facce). Questo avviene perché, non considerando per forza la geometria, il legame tra un bone ed un vertice è sempre definibile e calcolabile. La stessa cosa non vale per altri metodi, a meno che non si sfruttino opportune strategie, come ad esempio la voxellizzazione (in realtà in certe implementazioni degli altri metodi viene usata, ma non in modo mirato a superare questo problema, tant'è che in tali implementazioni vengono comunque richieste mesh completamente connesse).

2.1.3 Distanza Euclidea + euristiche

Per ovviare a questi inconvenienti in genere gli sviluppatori ricorrono ad euristiche per correggere almeno gli errori più evidenti, che comunque migliorano la situazione solo superficialmente.

In "Local Volume Preservation for Skinned Characters" (Rohmer, et al., 2008) ad esempio si utilizza il seguente approccio. Intanto per calcolare il peso di un bone *j* su un vertice *i*, si sfrutta la seguente formula:

$$w_{ij} = \left(\frac{l_j}{d_{ij}}\right)^{\alpha}$$

dove d_{ij} è la distanza Euclidea tra il bone *j* ed il vertice *i*, l_j è la lunghezza del bone ed α è un valore tale che $\alpha > 0$. Il metodo è quindi parametrizzabile tramite α.

Per correggere eventuali assegnamenti anatomicamente sbagliati si resetta il peso a 0 nel caso in cui il raggio r_{ij} tra il vertice *i* e il bone *j* su cui è calcolata la minima distanza Euclidea d_{ij} esca dalla mesh.

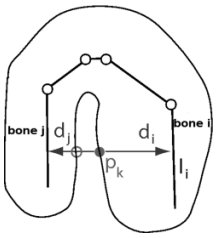


Figura 2.6: *il raggio dal vertice* p_k *al bone* j *esce dalla mesh, quindi il peso di* j *per* p_k *viene resettato a* 0.

Per stabilire quando ciò avviene si potrebbe usare un algoritmo di *ray-tracing*[7] che però risulterebbe pesante. In alternativa al ray-tracing gli autori utilizzano un'approssimazione confrontando il prodotto scalare tra il raggio r_{ij} e la *normale*[8] alla superficie. Quando l'angolo tra queste due direzioni è troppo piccolo si assume che il raggio esce dal dominio del corpo ed il valore di w_{ij} viene resettato a 0.

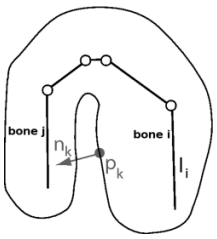


Figura 2.7: *utilizzo della normale* n_k *per stabilire se il raggio dal vertice* p_k *al bone* j *esce.*

Con quest'approccio (che sia basato su ray-tracing o su normale) possono comunque presentarsi vertici non raggiungibili con nessuno dei raggi minimi, si pensi ad esempio ad un vertice su un lobo di un orecchio. In questo caso si utilizza un'euristica: si considerano tali vertici della mesh come parti rigide e dunque si setta ad 1 il peso di skinning associato con il bone più prossimo. Infine tutti i pesi di uno stesso vertice vengono normalizzati a 1.

[7] Il ray-tracing è una tecnica di calcolo di un percorso in un ambiente ed è solitamente utilizzato per tracciare i raggi della luce. Nel nostro caso il percorso che ci interessa tracciare è definito dal raggio minimo tra il vertice e il bone. Solitamente tale algoritmo è abbastanza costoso.

[8] La normale è un attributo della superficie che specifica in che modo la luce viene riflessa sulla superficie stessa. Può quindi essere usata per stabilire qual è il versante della superficie rivolto verso l'esterno del volume.

Tuttavia la correzione effettuata può comunque dare dei risultati poco uniformi nel caso di skin molto rugosa. In questi casi infatti le normali di vertici vicini tra loro possono anche avere direzioni molto diverse e quindi coinvolgere bones diversi. Questo si è dimostrato quindi un approccio efficace nel caso di superfici morbide ed uniformi, ma non per superfici irregolari. In tali casi la correzione dovrebbe essere effettuata a mano, oppure bisognerebbe applicare l'algoritmo su una versione semplificata della skin. Inoltre come soluzione si tiene conto dell'anatomia, ma solo per l'esclusione di bones non coinvolti. Il metodo continua quindi ad avere un'individuazione delle fasce di massima deformazione non anatomicamente ottimale.

2.2 Distanza geodetica

Questo metodo verrà approfondito più avanti, per questo ora diamo solo un'infarinatura dei principi su cui si basa. Come descritto in "Atlas-Based Character Skinning with Automatic Mesh Decomposition" (Lu, et al., 2008), tale metodo sfrutta la distanza geodetica ispirandosi alla nozione di *Atlante di Carte*. Un *atlante* in topologia descrive come uno spazio complesso, nel nostro caso una mesh, sia formato da pezzi più semplici, detti appunto *carte*. Questo approccio si basa quindi su un atlante, cioè una segmentazione o *decomposizione* della mesh in parti chiamate carte (o anche *patches*, *pezze* o come le chiameremo noi *regioni*) disconnesse. La decomposizione è fatta sulle joints dello skeleton: tramite le joints si ritaglia la mesh e si dividono i vertici nelle varie patches.

Una volta segmentata la mesh abbiamo i *boundaries* (*bordi*, *confini*) di ogni regione. Partendo da questi bordi possiamo muoverci sulla superficie della mesh e raggiungere tutti i vertici calcolando la distanza geodetica. Questa è la minima distanza necessaria a raggiungere due punti muovendosi e scorrendo sopra una superficie. Proprio grazie a ciò questo metodo tiene conto intrinsecamente della geometria e dell'anatomia del modello ed evita i problemi della distanza Euclidea.

Ovviamente il costo è maggiore rispetto ai metodi basati su distanza Euclidea, ma ha comunque costi accettabili e la semplicità è notevole. Dunque è un buon compromesso tra bontà dei risultati e semplicità.

2.3 Diffusione del Calore

Un metodo che sta riscontrando molto successo si basa su un modello di diffusione del calore volumetrico (*volumetric heat diffusion*). Questo metodo utilizza un sistema di temperatura sfruttando l'analogia con l'equilibrio del calore per trovare i pesi.

Figura 2.8*: la figura sopra rappresenta l'equilibrio del calore per due bones; la figura sotto mostra i risultati dopo una deformazione*.

Praticamente il volume del personaggio viene trattato come un corpo conduttore di calore isolato ed ogni bone come un filamento che genera calore. Si forza quindi la temperatura su un bone *j* ad 1, mentre per tutti gli altri bones viene posta a 0. Dopo di ciò si diffonde il calore attraverso il modello.

La diffusione può essere realizzata ad esempio sul modello voxellizzato per facilitare l'operazione (Rosen, 2009).

Figura 2.9: *la mesh viene approssimata tramite una serie di cubi di dimensione identica che riempiono completamente il volume (voxellizzazione).*

Per ogni voxel impostiamo il calore con una media pesata tra il calore dei suoi vicini. Ripetiamo il procedimento fino a quando il calore non si diffonde completamente. Dunque per ogni vertice *i* prendiamo come peso di *j* la temperatura di equilibrio sulla superficie. Alla fine la distribuzione del calore è proporzionale al percorso più breve dal bone ad un qualsiasi punto del modello. In questo modo possiamo ottenere i pesi per ogni vertice confrontando il calore da ogni bone ad ogni vertice.

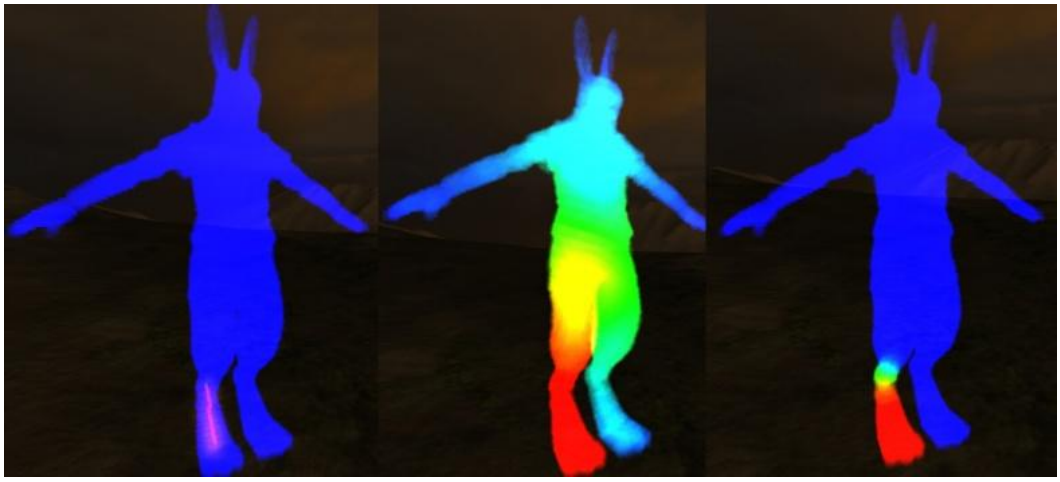


Figura 2.10*: la figura mostra come il calore si diffonda dal bone a tutto il volume.*

2.3.1 Pinocchio

Una tassellatura completa del volume può rallentare il procedimento. Per questo motivo un' implementazione dell'heat diffusion in una libreria chiamata Pinocchio, risolvone l'equilibrio solo sopra la superficie (Baran, et al., 2007), ma ad alcuni vertici aggiunge poi il calore trasferito dal bone più vicino. L'equilibrio sopra la superficie per il bone *j* è dato dalla formula $\frac{\partial \mathrm{w}^j}{\partial t} = \Delta w^j + \mathbf{H}(p^j - w^j) = 0$ riscrivibile come:

$$-\Delta w^j + \mathbf{H} w^j = \mathbf{H} p^j$$

Dove:

- Δ è la superficie Laplaciana discreta calcolata con la formula della cotangente (Meyer, et al., 2003)
- p^j è un vettore tale che: $p_i^j = 1$ se *j* è il bone più vicino al vertice *i*, e $p_i^j = 0$ altrimenti
- *H* è la matrice diagonale con $\mathbf{H}_{ii}$ rappresentante il peso del contributo di calore del bone più vicino al vertice *i*.

Siccome Δ ha unità di lunghezza^{-2}, lo stesso vale per **H**.

Per ogni vertice *i* viene quindi calcolata la distanza da ogni bones e si prende la minima *d*(*i*), identificando così anche quale sia il bone più vicino. A questo punto si utilizza un campo di distanza (calcolato in precedenza per effettuare anche il rigging) per stabilire se il segmento dal vertice al bone più vicino è completamente all'interno del volume del personaggio. Se ciò avviene si prende la distanza dal vertice *i* al bone più vicino *d*(*i*) e si pone $\mathbf{H}_{ii} = c/d(i)^2$, mentre si prende $\mathbf{H}_{ii} = 0$ se il segmento passa al di fuori della mesh. Il termine *c* è una costante per la quale, per valori del tipo $c \approx 0.22$, il metodo ritorna pesi con transizioni simili a quelle calcolate trovando l'equilibrio sul volume. Pinocchio usa $c = 1$, corrispondente ad una diffusione di calore anisotropa, ossia che segue la direzione del bone. In questo modo i risultati

sembrano più naturali. Quando *k* bones sono equidistanti dal vertice *j*, vengono usati tutti i loro contributi di calore: $p^j = 1/k$ per ciascuno dei *k* bones e si ha $\mathbf{H}_{ii} = k \cdot c/d(i)^2$.

L'equazione del calore sopra definita è un sistema lineare sparso e osserviamo che la matrice $-\Delta + \mathbf{H}$ a sinistra non dipende dal bone *j* per il quale stiamo calcolando il peso. Per questo si può fattorizzare il sistema una volta per tutte e sostituire all'indietro per trovare i pesi per ogni bone. Come descritto in "Efficient linear system solvers for mesh processing" (Botsch, et al.) si può usare un risolutore sparso di Cholesky[9] per calcolare la fattorizzazione per questo tipo di sistema. Pinocchio sfrutta la libreria TAUCS (Toledo, et al., 2003) per effettuare tale calcolo. Si noti che per ogni vertice la somma dei pesi w^j da 1:

$$(-\Delta + \mathbf{H}) \sum_j w^j = \mathbf{H} \cdot 1 \quad \Rightarrow \quad \sum_j w^j = 1$$

Gli autori aggiungono anche che il metodo è leggermente velocizzabile trovando i vertici che sono non-ambiguamente attaccati ad un unico bone e forzando il loro peso ad 1. Tuttavia asseriscono anche di aver implementato una variante di questo genere in precedenza, ma che tale soluzione portava miglioramenti trascurabili, tra l'altro introducendo occasionalmente artefatti.

Questo metodo dà dei risultati molto buoni, è assai più affidabile del metodo della prossimità dei vertici basato su distanza Euclidea e fornisce distribuzioni di peso lisce e realistiche. Di contro si ha una maggiore complessità nell'implementazione e nel costo. Inoltre, poiché la diffusione del calore in Pinocchio parte dai bones, diventa

[9] In generale il metodo di Cholesky viene utilizzato per risolvere sistemi lineari descritti tramite matrici simmetriche e definite positive. Un risolutore sparso di Cholesky ottimizza questo metodo alla trattazione di matrici sparse (Franco, 2009).

preponderante l'anatomia dello skeleton invece di quella del modello. Di conseguenza le fasce di massima deformazione non sono sempre ottimali e in certi casi si perde parte dell'anatomia della mesh.

2.4 Addestramento basato-su-esempi

Questo metodo è descritto in molti articoli [(Merry, et al., 2003), (Mohr, et al., 2003), (Wang, et al., 2002), (Jacka, et al., 2007)]. Rispetto agli altri metodi esaminati ha un approccio totalmente diverso al problema. Sfrutta infatti un database pre-calcolato di esempi creati manualmente da un artista o generati automaticamente da un sistema di modellazione basato-sulla-fisica, come quello sfruttato da Kry, James e Pai (Kry, et al., 2002). Come sappiamo tali metodi danno risultati di altissima qualità, ma il loro costo a run-time sarebbe esagerato. Tuttavia questi modelli ci servono solo per generare alcune pose che useremo per trovare i pesi; una volta stabiliti i pesi, sia i modelli del database che i metodi usati per generarli vengono completamente dimenticati e si utilizza l'LBS classico (o una sua estensione).

Il database è un set di mesh in alcune posizioni che rappresentano in qualche modo il risultato ottimale che vorremmo ottenere quando applichiamo l'LBS. Questo è quindi un approccio guidato dal risultato. Quello che si cerca di fare è "addestrare" l'LBS impostando i pesi in modo che forniscano il fit geometrico più vicino possibile al set di esercitazione di pose d'esempio. In pratica si prende una posa d'esempio dal database, si applica l'LBS per mettere la posa di riposo nella stessa posizione di quella d'esempio, dopo di che si guarda di quanto i vertici tra le due pose non coincidono memorizzando le differenze dei vettori e infine si tenta di correggere l'errore, ad esempio impostando un sistema di equazioni. Risolvendo quindi il sistema troviamo i pesi. Tale sistema avrà probabilmente più condizioni che variabili e può quindi essere

risolto tramite un solutore di minimi-quadrati[10] per fornire un'approssimazione che dia risultati più vicini agli esempi.

Un problema di questo metodo si evidenzia soprattutto quando il numero dei pesi è grande. Può capitare infatti che ad alcuni pesi venga data poca considerazione dalle pose esemplari. Ad esempio questo può verificarsi quando vi sono pose d'esempio in cui il movimento di un giunto giace su un piano. In tal caso il modello che dà il fit più vicino al set di esempi potrebbe causare problemi quando l'articolazione si muove per tutta la sua ampiezza. Per superare questo problema si introducono dei parametri che regolano in modo diverso i valori dei pesi non determinati e di quelli determinati.

La qualità è indubbiamente alta ed è possibile ottenere anche effetti secondari. Il problema principale è che le pose d'esempio devono essere scelte con cura per evitare problemi. Inoltre il metodo non è scalabile e i tempi per calcolare i pesi e la qualità dei risultati crescono col numero di pose d'esempio del database.

2.5 Tool per lo skinning

Esistono anche software dedicati ad effettuare lo skinning ed anche i più famosi programmi di modellazione (3D Studio Max, Maya e Blender) hanno propri metodi di generazione (semi-)automatica dei pesi. In tutti questi software spesso le operazioni di skinning sono abbinate anche al rigging.

[10] Il metodo dei minimi quadrati è metodo che permette di trovare una funzione, nota come *curva di regressione*, che il più vicina possibile ad un insieme di dati (tipicamente punti del piano). Tale funzione minimizza la somma dei quadrati delle distanze tra i dati la curva della funzione stessa.

2.5.1 La DemoUI di Pinocchio

Pinocchio fornisce anche una piccola applicazione chiamata DemoUI e sviluppata per testare i risultati dell'heat diffusion implementata all'interno di Pinocchio stesso.

I parametri della demo, per esempio la mesh e l'animazione, vengono inseriti tramite applicazione console. Il primo parametro deve essere la mesh con il suo path. Per specificare un'animazione bisogna usare l'opzione `-motion` seguita dal file dell'animazione con il suo path. Un esempio base di comando è il seguente:

```
DemoUI data/test.obj -motion data/walk.txt
```

Dove `test.obj` e `walk.txt` sono file forniti con la demo stessa, ma ovviamente possiamo usare qualsiasi mesh e animazione nei formati accettati da Pinocchio e che rispettino le caratteristiche richieste.

Le animazioni di Pinocchio sono file di testo con dati specificati in un formato interno di Pinocchio. L'opzione `-rot x y z degrees` permette di ruotare la mesh sugli assi prima di trattarla. L'opzione `-mo` visualizza solo la mesh senza alcuna analisi. Per specificare anche un proprio skeleton bisogna usare l'opzione `-skel` seguita dal file dello skeleton con il suo path. I file di skeleton devono essere testuali: ogni riga del file contiene l'indice del joint, le sue coordinate x,y,z e l'indice del joint padre. Se si è sicuri che lo skeleton è già ben posizionato all'interno della mesh si può utilizzare l'opzione `-nofit`: in tal modo Pinocchio genererà solo i pesi dei bones.

Una volta lanciato il comando si apre una finestra OpenGL in cui sono mostrate le animazioni. Con il tasto sinistro del mouse si può traslare la camera, la rotella del mouse permette di zoommare ed infine il tasto destro del mouse permette di ruotare la visuale. Tramite tastiera possiamo cambiare alcuni parametri di rendering: il tasto F

abilita/disabilita il *flat shading* e *smooth shading*[11], il tasto S mostra/nasconde lo skeleton, il tasto G mostra/nasconde il pavimento, il tasto T resetta la posizione della camera, il tasto Z allinea la visuale della camera al terreno.

La demo produce poi un file skeleton.out contenente lo skeleton posto all'interno della mesh ed un file attachment.out contenente i pesi calcolati dei bones. Entrambi i file sono apribili con un qualsiasi text editor. In skeleton.out lo skeleton è memorizzato nel formato descritto in precedenza con l'opzione `-skel`. In attachment.out ogni riga corrisponde ad un vertice della mesh e contiene i pesi per ogni bones, secondo l'ordine numerico che hanno i joints nello skeleton.

Oltre a ciò la demo contiene anche un paio di mesh d'esempio per testare i risultati ed altre mesh sono scaricabili sempre dal sito ufficiale.

2.5.2 Mixamo

Mixamo (Krasner, et al.) è un sito che fornisce un software on-line di rigging e skinning di una mesh. I tutorials e le guide in-linea di utilizzo del software presenti nel sito sono semplici e intuitivi. Inoltre ci sono molti video-tutorials che spiegano come sfruttare i risultati con tool di modellazione quali Blender, 3D-Studio, Maya e molto altro ancora.

Dopo una registrazione gratuita al sito e l'istallazione di un plugin gratuito (UnityWebPlayer.exe) che permette la visualizzazione 3D all'interno di un browser, è possibile caricare la propria mesh. Questa deve avere una dimensione massima di 30

[11] Il flat shading è il metodo più veloce e di qualità più bassa di colorazione di una superficie. Tale metodo colora le facce della mesh sfruttando l'angolo tra la normale della faccia e la direzione della sorgente luminosa, i loro colori e l'intensità della luce emessa. Con il flat shading diventano evidenti le facce e gli edge della superficie. I metodi più avanzati effettuano uno smooth (smussamento) per ottenere risultati visivi più morbidi.

MB, il personaggio deve essere biped (il che significa con due mani, due piedi, e una testa) e per ottenere dei buoni risultati deve essere in T-pose o A-pose.

La mesh può anche essere caricata senza skeleton nei formati .fbx, .obj o anche in un .zip che può includere a sua volta file .obj, .mtl e texture. In tal caso verrà richiesto di inserire manualmente 8 joints: il mento, i 2 polsi, i 2 gomiti, le 2 ginocchia e l'inguine. Da questi vengono poi generati automaticamente molti altri joints (in totale un po' meno di una trentina). In alternativa si può fornire, assieme alla mesh, uno skeleton nei formati fbx, .bvh e .zip (che include .fbx o .bvh più le texture). Il tempo di elaborazione di rigging e skinning è circa un paio di minuti.

Tralasciamo i passaggi intermedi per ottenere il risultato, rimandando al sito stesso per ulteriori approfondimenti e concentrandoci su quale potrebbe essere il metodo di skinning sfruttato. Non abbiamo trovato purtroppo una descrizione del metodo utilizzato, quindi ci siamo basati su un'analisi dei risultati per alcune mesh, per comprenderne il funzionamento ed osservare pregi e difetti. Da un'osservazione delle animazioni fornite dal tool abbiamo notato la presenza di perdite di volume tipiche dell'LBS, quindi possiamo supporre che sia questo l'algoritmo di animazione usato.

Figura 2.11: *mixamo presenta perdite di volume tipiche dell'LBS; si osservi ad esempio il gomito destro.*

Quando gli si passa una figura umana di corporatura molto standard (non troppo grassa e con i vari arti ben lontani tra loro) gli effetti sembrano molto buoni. Il metodo usato da Mixamo soffre di un appiattimento nella zona inguinale, presente come vedremo anche in Pinocchio, ma sembra un po' meno marcato.

Figura 2.12: *in mixamo si manifesta un forte appiattimento dell'inguine.*

I risultati comunque ad un primo impatto sembrano abbastanza soddisfacenti. Tuttavia se si tenta di trattare una figura umanoide più grassa o comunque più particolare, il risultato presenta i problemi tipici della distanza Euclidea che dà influenza maggiore anche a bones anatomicamente sconnessi dal vertice.

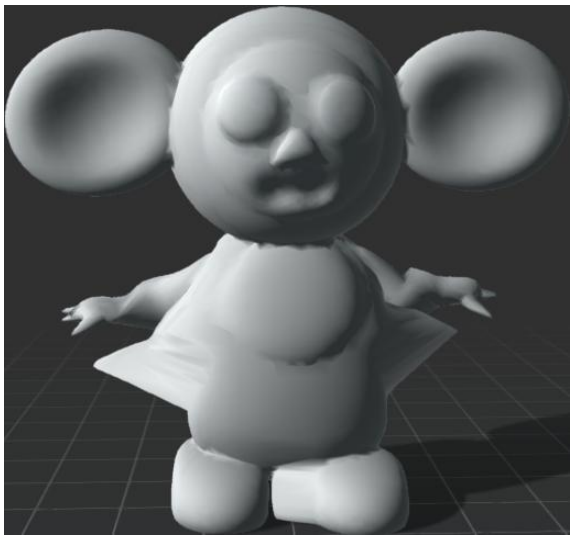


Figura 2.13: *Mixamo presenta il più evidente difetto dei metodi basati su distanza Euclidea.*

Oltre a ciò abbiamo testato che il software funziona anche con mesh formate da diverse sotto-mesh totalmente separate tra loro, il che, ricordiamo, è uno dei pochi vantaggi dei metodi basati su distanza Euclidea: non considerando la geometria, il legame tra un bone ed un vertice è sempre calcolabile (d'altro canto questo è lo stesso principio che genera i principali svantaggi della distanza Euclidea).

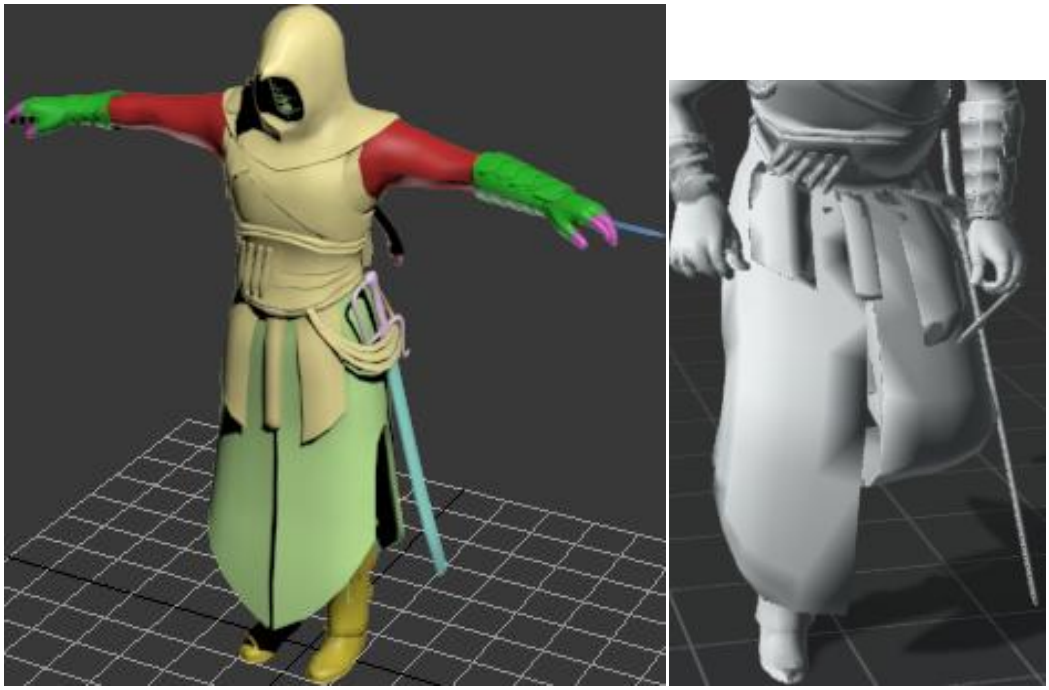


Figura 2.14*: la mesh è in realtà composta da tante diverse sottomesh separate, con nessun vertice in comune. Mixamo riesce comunque a trattarla. Si noti come il pugnale sulla mano sinistra (anch'esso composto da una mesh separata), viene deformato. Osserviamo anche che sul fodero della spada al fianco sinistro viene mantenuta una certa rigidità, quindi, probabilmente, per parti molto lontane la rigidità viene in qualche modo mantenuta.*

Quindi, anche se non ne abbiamo la certezza assoluta, abbiamo buone ragioni di credere che il metodo utilizzato da mixamo per stabilire i pesi per l'LBS sia basato su distanza Euclidea e che per figure umane abbastanza standard riesce a fornire risultati molto buoni grazie anche al fatto di utilizzare molti bones.

2.5.3 Blender

La versione 2.69 di Blender contiene un'add-ons, già interna alla versione base anche se disabilitata di default, per effettuare rigging e skinning in maniera semi-manuale. Dalla wiki ufficiale di blender (BlenderEN) è possibile consultare un tutorial, disponibile anche in italiano (BlenderIT), che ne spiega l'utilizzo. Si trovano anche molti video-tutorials che ne spiegano dettagliatamente l'utilizzo (Trammell).

Questa add-ons, dopo aver effettuato il rigging, si occupa di partizionare i vertici in gruppi secondo i bones. Per fare ciò sfrutta la distanza dai bones e difatti per un dato gruppo possono essere selezionati diversi vertici scorretti. Il modellatore deve quindi correggere questi errori a mano escludendo i vertici dal gruppo. Una volta ripulite tutte le partizioni il modello è già pronto e i pesi sono già definiti.

Questo metodo probabilmente assegna pesi partendo dai gruppo creati, decrementando il peso man mano che ci si allontana e normalizzando poi i pesi tra tutti i bones coinvolti. È quindi verosimilmente un metodo basato su distanza Euclidea che sfrutta delle capsule generate in modo semi-automatico.

Su internet si trovano anche riferimenti ad un'implementazione della diffusione del calore per Blender, ma il link ufficiale non è più attivo e fin'ora non abbiamo trovato alternative.

2.5.4 3D Studio Max

Per quanto riguarda 3D Studio esistono diversi script [(super_oko) e (kogen)] disponibili che effettuano l'assegnazione automatica dei pesi, assieme ad altrettanti video-tutorials. Questi script funzionano in maniera molto simile all'add-ons di Blender, partizionando la mesh e utilizzando la distanza dalle partizioni per distribuire i pesi.

2.5.5 Maya

In Maya si possono utilizzare ben tre metodi di calcolo dei pesi di skinning: *Closest distance* basato su distanza Euclidea, *Closest in hierarchy* che sfrutta ancora la distanza Euclidea ma tenendo conto della gerarchia dello skeleton ed infine *Heat Map* che sfrutta un'implementazione del metodo basato su diffusione del calore.

Per effettuare il calcolo dei pesi bisogna selezionare dal menù a tendina in alto la voce Skin, quindi Bind Skin e infine Smooth Bind. Si aprirà così la finestra di dialog Smooth Bind Options. Da tale finestra nel campo Bind method si può scegliere tra i 3 metodi Closest distance, Closest in hierarchy e Heat Map. Segnaliamo anche che subito sotto, tramite il campo Skinning method, è possibile scegliere l'algoritmo di skinning tra Classic linear che è l'LBS classico, Dual Quaternion che sfrutta i quaternioni duali e Weight blended che permette di impiegare una fusione tra l'LBS classico e quello con quaternioni utilizzando una mappa sui pesi definita dall'utente.

Esiste anche una guida di riferimento riguardante la dialog Smooth Bind Options (Autodesk, 2014) ed abbiamo già citato anche un video tutorial (Roselle, 2012) che ne mostra l'utilizzo. Abbiamo anche trovato un'implementazione del metodo di Pinocchio in uno script per Maya, ma supportato solo fino alla versione 2010 del programma (barnabas79, 2010).

2.6 Scelta del metodo da implementare

Per i molti problemi noti abbiamo scartato l'idea di utilizzare metodi basati sulla distanza Euclidea. Infatti essendo le mesh acquisite in-line non vi è garanzia della loro bontà a meno di pre-elaborazione, ad esempio da un punto di vista della rugosità o della posizione.

Come già accennato abbiamo trovato una libreria open source chiamata Pinocchio che, oltre ad occuparsi del rigging, implementa anche il metodo basato sulla diffusione del calore. Per questo abbiamo preferito non re-implementare un altro metodo di heat diffusion, ma al limite integrare il nostro con quello di Pinocchio in una demo per comparare i risultati.

Per quanto riguarda l'addestramento basato su esempi, il problema principale è che le pose d'esempio vanno scelte con criterio. Nel nostro caso, poiché cerchiamo un metodo completamente automatico, non abbiamo garanzie che le pose scelte siano le migliori possibili. Inoltre preferiremmo un metodo che non impieghi un tempo troppo eccessivo. Nell'addestramento infatti, oltre al fatto che il tempo di calcolo dipende dalla quantità di pose d'esempio generate, è soprattutto il generare tali pose che porterebbe via molto tempo. Oltre al fatto che dovremmo implementare da zero un metodo basato sulla fisica che generi le pose d'esempio, il che comporterebbe un ulteriore lavoro decisamente gravoso.

Si è quindi deciso di implementare il metodo della distanza geodetica (Lu, et al., 2008), integrandolo nella demo di Pinocchio e sviluppando la demo stessa, di modo da poter confrontare i risultati derivati dall'applicazione dei due metodi. In realtà, come vedremo, gli stessi autori di questo metodo in un lavoro successivo (Hétroy, et al., 2009) lo hanno generalizzato, modificandone alcune parti e rendendolo adattabile anche all'utilizzo di un qualsiasi metodo di distanza, comprese la distanza Euclidea e la distanza armonica (ovvero la diffusione del calore). Noi, tuttavia, nel nostro tool ci limiteremo all'implementazione tramite distanza geodetica per i motivi sopra citati.

Capitolo 3
L'algoritmo di skinning automatico Atlas-Based

Abbiamo detto che tra i vari metodi per calcolare i pesi scegliamo quello che calcola la distanza geodetica basandosi su Atlante di Carte (Lu, et al., 2008). Di seguito, come prima cosa delineeremo i passi base dell'algoritmo, quindi vedremo una sua generalizzazione (Hétroy, et al., 2009), analizzeremo poi alcuni dei passi e infine, sulla base di tali analisi, cercheremo dei possibili miglioramenti all'approccio.

3.1 Overview

Il metodo che andiamo a descrivere comincia con una decomposizione della mesh in *regioni* topologicamente equivalenti a cilindri e legate ad uno o più bones dello skeleton. Una regione può anche essere vista come una sottomesh, cioè un sottoinsieme delle facce e dei vertici della mesh di partenza. In particolare l'unione di

tutte le sottomesh coincide con la mesh stessa. La mesh viene così spezzata in un certo numero di parti secondo gli arti. Il secondo passo estende queste regioni vicino ai joints. Così facendo le *regioni estese* avranno parti che si sovrappongono tra loro[12]. Tali parti sono dette *aree di sovrapposizione*. Anche le regioni estese sono sottomesh, ma in questo caso l'unione delle sottomesh porta ad una duplicazione di alcune facce: quelle che si trovano nell'area di sovrapposizione.

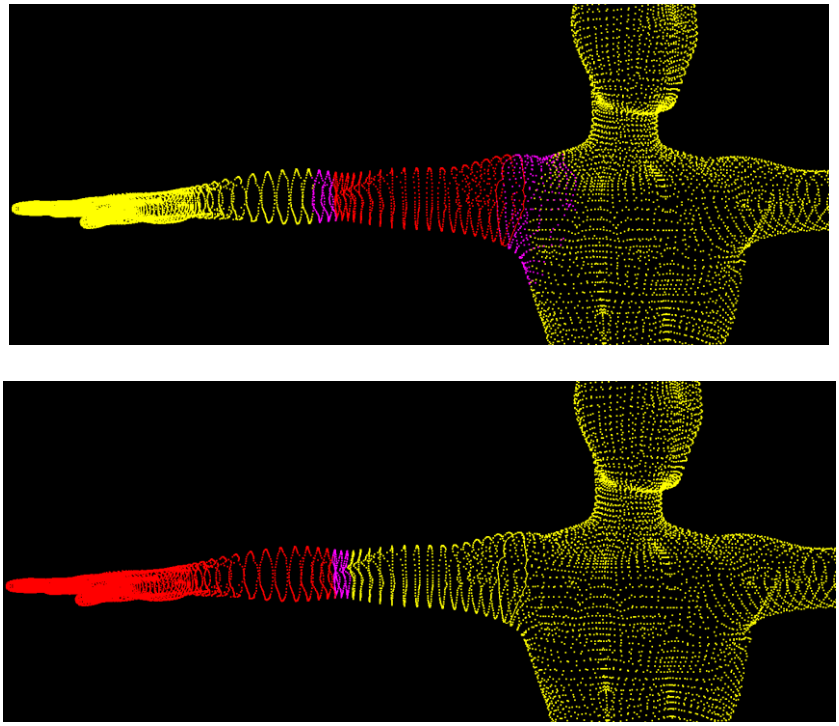


Figura 3.1*: nelle due immagini l'insieme dei vertici rossi identifica una regione; mettendo assieme i vertici fucsia sul gomito delle due immagini si identifica un'area di sovrapposizione, mentre una regione estesa è costituita dai vertici rossi assieme a quelli fucsia in una singola immagine. Si noti come le regioni rosse delle due immagini possono essere riunite senza considerare due volte uno stesso vertice.*

Nel terzo passo si procede con il calcolo vero e proprio dei pesi per ogni vertice utilizzando la distanza geodetica dai vertici sino ai bordi delle aree di sovrapposizione.

[12] I concetti di regione e regione estesa possono anche essere ricondotti ai concetti rispettivamente di capsula interna e capsula esterna viste nel paragrafo 2.1.1 .

Uno schema generale del procedimento è il seguente[13]:

1. Generazione delle regioni
   1.1. Individuazione dei confini tra due regioni
   1.2. Individuazione delle regioni a partire dai confini
2. Estensione delle regioni o generazione delle aree di sovrapposizione. Per ogni boundary:
   2.1. Calcolo della distanza geodetica da ogni vertice ad ogni confine
   2.2. Calcolo della lunghezza di sovrapposizione centrata nel confine
   2.3. Generazione dell'area sovrapposta segnando per ogni vertice tutte le regioni cui appartiene, con la relativa distanza geodetica da queste.
3. Definizione dei pesi per ogni bone sfruttando la distanza geodetica calcolata in precedenza

Il metodo proposto è inoltre pensato per sfruttare informazioni semantiche attaccate allo skeleton, per avere una pesatura più realistica.

3.2 Fase 1: decomposizione della mesh

Il primo passo consiste quindi nel decomporre la mesh in regioni sfruttando lo skeleton. In altre parole si segmenta la mesh spezzandola usando alcuni joints dello skeleton. Per una panoramica dei metodi di decomposizione si veda l'APPENDICE B.

Il metodo che utilizziamo è un approccio implicito basato-su-skeleton (secondo la classificazione data in APPENDICE B). Utilizzando uno skeleton che abbia anche informazioni anatomiche aggiuntive si può potenziare molto questa decomposizione

[13] Quelli che qui sono il primo e secondo passo nel documento originale sono visti come due sottopassi di un unico passo. Qui preferiamo vedere il procedimento in due passi principali per dare poi una visuale più globale delle sottofasi.

automatica. Il metodo suggerito dagli autori per avere uno skeleton con informazioni anatomiche è descritto in "Harmonic skeleton for realistic character animation" (Aujay, et al., 2007). Tale metodo porta informazioni associate ai joints ed inoltre indica l'asse di simmetria del modello, ma si può anche utilizzare un qualsiasi skeleton con informazione aggiunta.

La segmentazione della mesh è calcolata in due fasi in modo rapido ed automatico.

3.2.1 Rilevamento del confine

Ogni *boundary* (*confine, bordo*) della decomposizione separa le regioni adiacenti. Questi confini dovranno corrispondere alle aree in cui la mesh si deforma maggiormente. Ad esempio un confine può essere attorno ad un gomito, un'altro sulla spalla ecc. Pertanto, ciascuno dei confini dovrebbe corrispondere ad un joints dello skeleton: per ogni confine c'è un joints che l'ha generato (ma, come vedremo più avanti, non è detto che ogni joints generi un confine). Si è scelto di definire i confini come curve chiuse (concatenazioni di segmenti) con lunghezza localmente minimale.

Ogni curva è generata tramite l'intersezione tra la mesh ed un *piano di taglio* che passa attraverso uno dei joint. Per facilitare la scelta del piano di taglio, questo viene calcolato per ogni joint con solo 2 bones incidenti, mentre i joints con più di due bones incidenti non sono valutati. In generale un piano nello spazio è definibile tramite un punto appartenete al piano e la normale al piano stesso. Per indicare quindi un piano di taglio possiamo utilizzare la notazione $P(J,n)$, dove n è il vettore normale al piano e J è un punto attraverso cui passa il piano, ovvero J è il joint da cui generiamo il piano.

Dato quindi un joint J con solo 2 bones incidenti dobbiamo trovare una normale n per poter definire completamente il piano $P(J,n)$. Il *bone superiore* di J è definito da J e dal suo joint padre nella gerarchia dello skeleton, mentre il *bone inferiore* di J è

definito da *J* e dal suo joint figlio. Notiamo che, poiché in *J* incidono solo 2 bones, nei casi che consideriamo *J* può avere un solo joint figlio (mentre è sempre valido che ogni joint ha un solo padre, essendo lo skeleton una gerarchia ad albero). Imponiamo quindi anche la condizione aggiuntiva che la normale *n* di $P(J,n)$ che stiamo cercando deve giacere sul piano *Q*, determinato dal bone inferiore e dal bone superiore di *J*. In altre parole *Q* è un piano che passa per i 3 joints coinvolti e appartenenti ai 2 bones interessati: cioè *Q* passa per *J*, per il joint padre di *J* e per il joint figlio di *J* (ed è facile calcolare il piano passante per 3 punti nello spazio). Grazie a questa condizione aggiuntiva possiamo determinare un insieme discreto di normali $\{n_1, n_2,\ldots, n_k\}$, ognuna delle quali definisce anche un piano $P_1, P_2,\ldots, P_k$, poiché per tutte queste normali abbiamo già un punto appartenente ai piani: infatti *J* è un punto comune a tutti i piani $P_1, P_2,\ldots, P_k$.

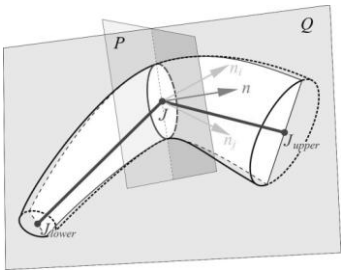


Figura 3.2: *si usa un piano di taglio P passante per il joint J per sezionare la mesh ed ottenere un bordo, indicato nella figura dal cerchio contenente J. Le normali (come n_i ed n_j) che generiamo devono giacere sul piano Q e tra queste scegliamo poi quella che crea il bordo più piccolo. Questa sarà la normale n del piano P che cerchiamo.*

Ora dobbiamo scegliere uno solo di questi piani per avere un unico piano $P(J,n)$ da associare a *J*. Per ogni piano P_i di un dato *J* effettuiamo quindi l'intersezione con gli edges della mesh: un edge interseca il piano quando i suoi vertici estremi si trovano in parti opposte rispetto al piano. Gli edge che intersecano il piano sono *edge di confine* o *edge di intersezione*. Per ogni edge di confine si genera esattamente un *punto di confine* tramite l'intersezione con il piano. Quando due edge di confine hanno un vertice estremo in comune significa che appartengono ad uno stesso triangolo (o faccia), che possiamo quindi chiamare *faccia di confine*, ed in tal caso

possiamo generare un segmento tra i punti di confine dei due edge di confine, detto *segmento di confine*.

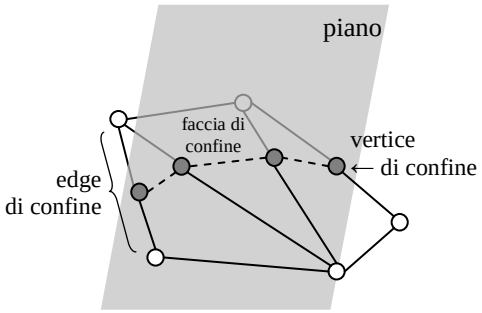


Figura 3.3: *la linea tratteggiata indica il confine che passa nei vertici di confine. Ogni vertice di confine è distinguibile dal suo interno grigio scuro. Ogni faccia di confine ha due edge di confine. La concatenazione dei vertici di confine è la stessa dell'adiacenza delle facce di confine ed è indicata dalla linea tratteggiata che è il confine stesso.*

Concatenando i segmenti di confine tramite i loro punti di confine si genera una sequenza chiusa di segmenti che giacciono sul piano P_i e sulla mesh. Tale sequenza è il bordo B_i del piano P_i. Dunque B_i rappresenta una curva chiusa che si trova sulla superficie della mesh. Si noti però che tagliando la mesh tramite un piano P_i si può generare più di una componente connessa chiusa. Tra queste componenti generate teniamo solo la curva chiusa al cui interno giace il joint *J*, scartando le altre ed ottenendo così un confine B_i formato da una sola sequenza chiusa per ogni piano P_i.

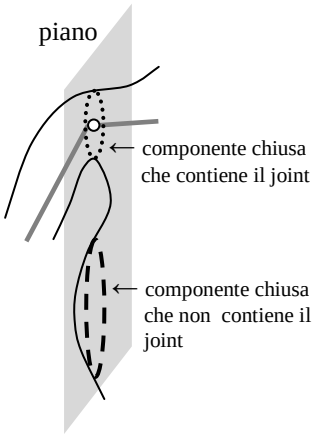


Figura 3.4: *il piano di taglio genera due componenti chiuse: una punteggiata ed una tratteggiata. Di tutte le componenti chiuse si tiene solo quella che contiene il joint: nell'immagine solo quella punteggiata.*

A questo punto per il singolo joint J abbiamo tanti boundary B_1, B_2,…, B_k: uno per ogni piano P_1, P_2,…, P_k. Tra questi ne dobbiamo scegliere solo uno per il joint J. Si valutano tutti i B_i generati con una funzione $F(B_i)$, e prendiamo il piano con $min(F(B_i))$. La funzione $F(B_i)$ può essere definita come il perimetro oppure l'area del confine B_i. Gli autori nella pratica hanno sfruttato il perimetro, che sembra avere un buon comportamento.

Otteniamo infine così un solo boundary per ogni joint che ha solo due bones incidenti.

3.2.2 Segmentazione

La segmentazione viene eseguita tramite aumento-della-regione. Una volta che abbiamo ottenuto tutti i confini B_i espliciti per ogni joint con incidenti 2 soli bones, è semplice segmentare la mesh in sotto-mesh (o regioni) tramite un algoritmo di aumento-regione. Per ogni confine, segniamo i triangoli di intersezione della mesh come facce di confine. Cominciamo quindi con una faccia a caso non-di-confine, e facciamo crescere una sotto-mesh incrementalmente aggiungendo ogni volta le facce adiacenti che sono non-di-confine. Una volta trovate tutte le facce adiacenti non-di-confine, tutte queste assieme formano una sotto-mesh, ovvero una regione. Procediamo quindi con un'altra faccia non-di-confine che non è stata ancora visitata per adiacenza ad un passo precedente. L'algoritmo termina quando tutte le facce sono state visitate. Ciascuna faccia di confine potrebbe essere assegnata a due regioni, mentre al termine del procedimento una faccia non-di-confine appartiene ad una ed una sola regione.

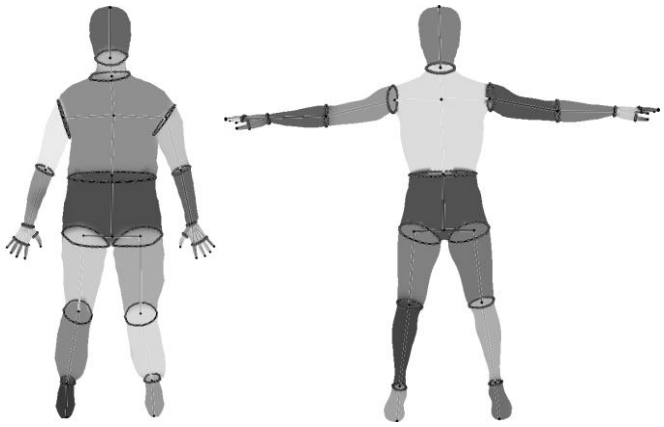


Figura 3.5: *segmentazione per due modelli umani. Ogni regione è indicata con una diversa tonalità di grigio così come i bones. Le curve di confine sono segmenti neri che uniscono i punti (vertici di confine).*

Gli autori suppongono di gestire solo modelli a 0-genus[14], quindi da $k - 1$ confini si otterranno k-regioni della mesh. Ciascuno dei confini ha due regioni adiacenti, il che dipende dall'anatomia delle mesh trattate e dal metodo di partizionamento, mentre ogni regione ha almeno un confine.

3.3 Fase 2: estensione delle regioni

In questa fase ogni regione viene estesa aumentandone i confini, cioè aggiungendo vertici ed edge esterni alla regione, partendo dai confini e muovendosi verso le regioni adiacenti. In questo modo si avranno delle parti nella regione estesa che appartengono anche alle regioni (estese) adiacenti. Queste parti sono dette *overlapping areas* (*aree di sovrapposizione*). Per estendere le regioni si sfrutta la distanza geodetica ed il diametro.

[14] Il genus per una superficie chiusa manifold è il massimo numero di tagli lungo curve chiuse semplici effettuabili senza sconnettere la mesh. Con *k*-genus si intende che si possono effettuare al massimo *k* tagli esenza sconnettere la mesh. Informalmente possiamo dire che *k* è il numero di maniglie o buchi di una mesh. Una sfera è un 0-genus, una ciambella è un 1-genus, una mesh rappresentante un 8 è un 2-genus, ecc.

3.3.1 Calcolo della distanza geodetica

Abbiamo detto di aver $k - 1$ confini $\{B_i\}_{i=1}^{k-1}$ e k regioni $\{R_j\}_{j=1}^{k}$. Inoltre ogni boundary B_i ha 2 regioni adiacenti indicate con R_{i1} e R_{i2} ed ogni regione R_j ha n confini $\{B_{jk}\}_{k=1}^{n}$.

Ora, per ogni vertice v che appartiene alla regione R_j, calcoliamo la distanza geodetica verso ogni boundary B_{jk} della regione. Dato un vertice e un boundary la distanza geodetica può essere calcolata trovando la minima distanza geodetica da v a tutti i punti di B_{jk}. Si genera in altre parole un campo di distanza per tutti i boundary.

L'algoritmo Atlas-based calcola la distanza geodetica da ogni vertice della mesh separatamente per ogni punto di tutte le curve di confine, dopo di che prende la distanza minima. Se il bordo B_i è composto da m punti $\{c_1, c_2, \dots , c_m\}$ vengono calcolate m distanze geodetiche: $g(v, c_j)$ è la distanza geodetica dal vertice v al punto c_j (per $j = 1,\dots,m$). Prendiamo quindi la minima tra queste distanze e la usiamo come distanza geodetica da v a B_i: $g(v, B_j) = \min g(v, c_j)$

Per velocizzare i conti gli autori prima trovano il punto c_k più vicino a v tramite la distanza Euclidea, dopo di che calcolano la distanza geodetica tra v e soltanto un numero fisso (per esempio 5) di punti di confine vicini a c_k.

Il calcolo effettivo di ogni distanza geodetica $g(v, c_j)$ avviene tramite un preciso metodo [(Tang, et al., 2003) o (Novotni, et al., 2002)]. Se la mesh è manifold, una rappresentazione topologica tramite half-edge (vedremo più avanti in cosa consiste questo tipo di rappresentazione) è particolarmente adatta al calcolo della distanza geodetica, poiché rende semplice l'individuazione del vertice opposto ad un edge entrambi appartenenti alla stessa faccia (basta usare l'attributo *prec* dell'half-edge) e permette inoltre di navigare facilmente tra le facce (tramite l'attributo *twin* dell'half-edge). Una spiegazione generale di questi algoritmi è data in APPENDICE C.

3.3.2 Calcolo delle aree di sovrapposizione

Adesso per ogni boundary B_i dobbiamo stabilire di quanto possiamo muoverci verso le due regione adiacenti al boundary stesso. Per ogni B_i possiamo calcolare il suo diametro approssimato D_i. Tale diametro indica, centrandosi in B_i, di quanto estendere l'area di sovrapposizione.

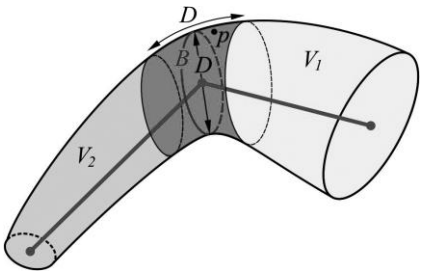


Figura 3.6: *V1 e V2 non sono proprio le regioni adiacenti al bordo B, ma sono le regioni private della parte che è finita nell'area di sovrapposizione; in grigio scuro è indicata l'area di sovrapposizione che si estende, centrandosi in B, di una distanza pari a D (o se preferiamo, partendo da B si estende di una distanza pari a D * 0,5 nelle due direzioni delle 2 regioni).*

La scelta di utilizzare il diametro approssimato di ogni confine per stabilire l'ampiezza dell'area di sovrapposizione non è obbligatoria: nella pratica si può utilizzare una qualsiasi distanza, anche diversa per ogni joint. Tuttavia è risaputo che questo criterio aiuta ad mascherare gli artefatti di collapsing elbow dell'LBS (Baran, et al., 2007).

Se definiamo inoltre le *aree di estensione*[15] come le due metà dell'area di sovrapposizione separate dal bordo, possiamo anche definire per ogni regione una sua *regione estesa*, che è pari alla regione stessa unita alle aree di estensione sui tutti i suoi bordi. Le regioni estese verranno usate nella successiva generalizzazione dell'algoritmo (Hétroy, et al., 2009). Notiamo che le aree di estensione vengono generate partendo dal bordo con una distanza pari al raggio, cioè a metà diametro.

[15] Le aree di estensione sono quelle che nella trattazione dei Sistemi Informativi Territoriali vengono chiamate aree di rispetto. Un'*area di rispetto* è una parte dello spazio geografico che circonda un'entità puntuale, lineare o areale. Nel nostro caso l'entità è di tipo areale: sono le regioni.

3.3.3 Generazione dell'area sovrapposta

Utilizziamo ora il diametro D_i e la distanza geodetica calcolata per generare concretamente la parte di sovrapposizione per le due regioni contigue R_{i1} e R_{i2} al relativo B_i. Per fare ciò segniamo che tutti i vertici con la distanza da B_i a meno di D_i * 0,5 fanno parte dell'area di sovrapposizione e quindi sono influenzate da tutte le regioni adiacenti a B_i. Come risultato di questa operazione ogni regione R_j con n confini avrà n parti sovrapposte.

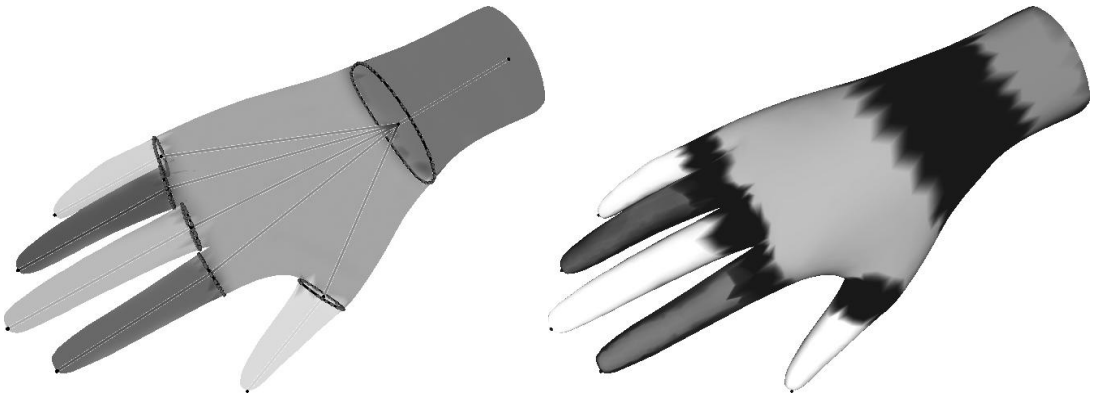


Figura 3.7: *a sinistra sono mostrate le regioni di partenza generate dalla segmentazione e i confini; a destra le aree di sovrapposizione calcolate a partire dai confini delle regioni sono colorate in nero.*

3.4 Fase 3: definizione dei pesi

L'ultimo passo mira finalmente al calcolo vero e proprio dei pesi. Per calcolare tali pesi ci serviremo di una funzione definita in 3 casi che, dato un vertice e due bordi, indica quanto il vertice è sbilanciato rispetto ai due bordi. Per fare ciò la funzione sfrutta la distanza geodetica dal bordo rispetto alla lunghezza del diametro del bordo stesso. I valori di questa funzione saranno poi trattati tramite un'altra funzione per renderli più morbidi rispetto al loro sbilanciamento.

3.4.1 La funzione β

Prima di definire i pesi introduciamo una funzione cubica β che soddisfa le condizioni di Hermite $\beta(0) = 0, \beta(1) = 1$ e $\beta'(0) = \beta'(1) = 0$. Tramite semplici calcoli si ottiene:

$$\beta(l) = -2l^3 + 3l^2$$

Questa funzione consente di definire pesi che diminuiscono in modo uniforme tanto più il vertice *v* è prossimo al confine della regione, fornendo risultati visivamente migliori. Sempre nella generalizzazione dell'algoritmo (Hétroy, et al., 2009) sono trattati i risultati derivati dall'utilizzo di una funzione lineare al posto di β.

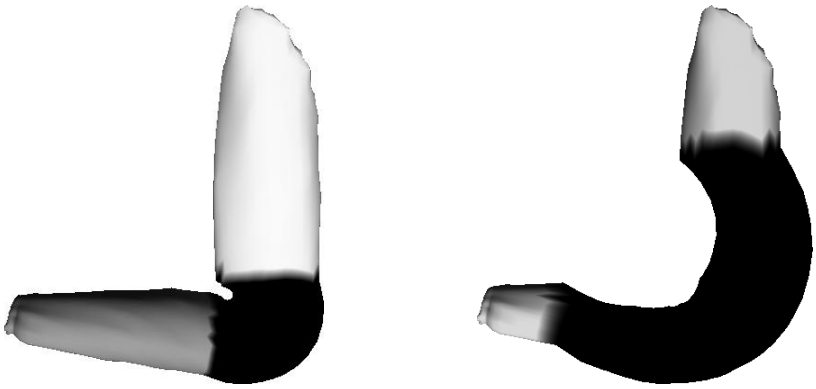


Figura 3.8: *nell'immagine a sinistra sono mostrati i risultati utilizzando una funzione lineare al posto di β. A destra si usa ancora la funzione lineare ed i problemi sono resi ancora più evidenti prendendo un valore di estensione delle regioni pari a* 2 * D_i *invece che* 0,5 * D_i.

Questi mostrano come l'utilizzo di una funzione lineare dia risultati molto meno naturali. Ciò avviene poiché vicino ai confini varia molto l'influenza dei bones. La funzione β, al contrario, ha la proprietà di aumentare molto lentamente per valori di l vicini a 0 e per valori vicini a 1 (cioè vicino ai contorni delle aree di sovrapposizione). Questa proprietà ci fornisce risultati visivamente migliori. Tale funzione è indipendente dall'articolazione. Tuttavia i pesi possono essere trattati con una qualsiasi funzione tale che $\beta(0) = 0$ e $\beta(1) = 1$ e magari che sia anche dipendente dall'anatomia dell'articolazione.

3.4.2 Calcolo dei pesi

Ricordiamo che una regione può essere formata da più bones. In questo algoritmo infatti le regioni hanno "sostituito" i bones originali. Dunque i pesi vanno calcolati per le nuove regioni e non per i bones originali. Prendiamo ora un vertice *v* e tutte le regioni $R_1,\ldots,R_n$ che lo contengono. Per ogni R_j abbiamo un bordo B_j, con $j \in \{1,\ldots,n\}$. Al passo precedente abbiamo calcolato la distanza geodetica da *v* ad ogni B_j. Nel caso particolare in cui $n = 1$, il vertice *v* appartiene esattamente ad una sola regione (R_1). In questo caso è facile stabilire che, per il vertice *v*, il peso di tale regione è 1, mentre per tutte le altre è 0. Se invece $n > 1$ il peso di w_j per *v* è dato da:

$$w_j = \frac{2}{n(n-1)} \sum_{i \neq j} \beta\left(\frac{d_j}{d_i + d_j}\right)$$

Si osservi che poiché $\beta(1-l) + \beta(l) = 1$ implica che:

$$\sum_{j=1}^{n} w_j = \frac{2}{n(n-1)} \sum_{1 \leq i < j \leq n} \left[\beta\left(\frac{d_j}{d_i + d_j}\right) + \beta\left(\frac{d_i}{d_i + d_j}\right)\right] = 1$$

Per esempio, nel caso particolare in cui *v* appartiene esattamente solo a 2 regioni, otteniamo:

$$w_1 = \beta\left(\frac{d_1}{d_1+d_2}\right) \qquad \text{e} \qquad w_2 = \beta\left(\frac{d_2}{d_1+d_2}\right) = 1 - w_1$$

Per ora abbiamo solo definito come dovrebbero essere i pesi. Passiamo al calcolo vero e proprio. Diamo prima però alcune definizioni. Le regioni estese a cui *v* appartiene sono dette *regioni relative*. Invece, se *v* appartiene ad una regione (e quindi anche alla sua regione estesa) indichiamo tale regione come *proto-regione* di *v*. In altre parole le regioni relative e le proto-regioni sono il corrispettivo delle regioni estese e delle regioni, ma riferite ad un preciso vertice *v*. È ovvio che ogni vertice ha

esattamente una sola proto-regione ed ha almeno una regione relativa. Se poi è un vertice che appartiene ad un'area sovrapposta avrà almeno 2 regioni relative.

Prendendo ad esempio un modello umano i vertici attorno al gomito, al ginocchio ecc. avranno 2 regioni relative, mentre alcuni vertici in prossimità del bacino avranno 3 regioni relative e per alcuni vertici sul petto (non considerando le clavicole) ci possono essere anche 4 regioni relative.

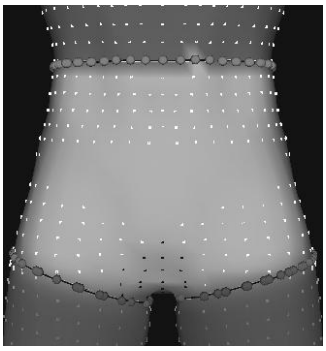


Figura 3.9: *nell'immagine i vertici bianchi hanno 2 regioni relative, mentre i neri hanno 3 regioni relative. Per i vertici neri la proto-regione è la regione più chiara.*

Abbiamo già stabilito come comportarci con i vertici non sovrapposti, cioè quelli con una sola regione relativa: questi vertici seguono solamente la trasformazione della loro regione relativa, nonché proto-regione, in modo totalmente rigido. Possiamo quindi concentrarci solo sui vertici sovrapposti, cioè quelli che hanno più di una regione relativa.

Per l'equazione che abbiamo definito, la prima cosa da fare è trovare tutte le distanze d_i tra v e ogni rispettivo R_i, con $i \in \{1,\ldots,n\}$. In una fase precedente dell'algoritmo, più precisamente quando abbiamo generato le aree di sovrapposizione partendo dalle curve di confine B_i, abbiamo costruito un campo di distanza geodetica da ogni vertice ad ogni curva di confine associata. Supponendo che la curva di confine tra R_i e R_j sia $B_{i,j}$, avremo tale campo ci fornisce la distanza geodetica $g(v,B_{i,j})$ da v a $B_{i,j}$. La lunghezza dell'area di sovrapposizione che abbiamo calcolato a partire da $B_{i,j}$ è $D(B_{i,j})$, semplificabile come $D_{i,j}$.

Possiamo quindi calcolare la distanza da un vertice *v* a un bordo $B_{i,j}$ come:

$$d_{i,j}(v) = \begin{cases} \dfrac{\frac{D_{i,j}}{2} + g\left(v, B_{i,j}\right)}{D_{i,j}} & \exists B_{i,j} \;\&\; R_i \text{ è } proto-regione \text{ di } v \\ \dfrac{\frac{D_{i,j}}{2} - g\left(v, B_{i,j}\right)}{D_{i,j}} & \exists B_{i,j} \;\&\; R_j \text{ è } proto-regione \text{ di } v \\ 0.5 & B_{i,j} \text{ non esiste} \end{cases}$$

$$i = 1,2, \dots n;\; j = 1,2, \dots n;\; j \neq i$$

A questo punto i pesi del vertice *v* possono essere calcolati tramite:

$$w_j = \frac{2}{n(n-1)} \sum_{i \neq j} \beta\left(d_{i,j}\right)$$

Prendiamo ancora $n = 2$ come esempio. Il vertice *v* avrà quindi due regioni relative R_1 e R_2. Supponiamo che R_1 sia la proto-regione di *v*: in tal caso R_1 e R_2 avranno ovviamente un bordo in comune, $B_{1,2} = B_{2,1} = B$, che avrà diametro $D_{1,2} = D_{2,1} = D$. I pesi delle due regioni per il vertice *v* saranno quindi:

$$w_1 = \beta\left(d_{1,2}\right) = \beta\left(\frac{\frac{D_{i,j}}{2} + g\left(v, B_{i,j}\right)}{D_{i,j}}\right)$$

$$w_2 = \beta\left(d_{2,1}\right) = \beta\left(\frac{\frac{D_{i,j}}{2} - g\left(v, B_{i,j}\right)}{D_{i,j}}\right) = 1 - w_1$$

Notiamo che nel caso si abbia una sola regione relativa (nonché proto-regione) nella funzione di calcolo dei pesi la sommatoria sarebbe nulla o se preferiamo non sarebbe applicabile. Per questo, come abbiamo detto, per i vertici con una sola regione relativa non usiamo tale funzione, ma li trattiamo a parte assegnando peso 1 alla loro unica regione e 0 a tutte le altre regioni.

3.5 Generalizzazione del metodo

Come abbiamo accennato questo metodo basato su segmentazione è stato rielaborato successivamente (Hétroy, et al., 2009), rendendolo più generale, in modo da poter utilizzare anche diverse funzioni per la distanza: Euclidea, geodetica o armonica. In pratica una volta stabiliti i bordi per le prime due fasi si procede in maniera analoga a quanto descritto in precedenza: per ogni regione si parte dai suoi bordi calcolando la distanza dai vertici esterni alla regione con una qualsiasi delle funzioni di distanza citate; da ogni regione si ottiene quindi una regione estesa aggiungendo i vertici delle regioni adiacenti che rientrano nell'area di estensione.

3.5.1 Generalizzazione con la distanza armonica

Per quanto riguarda l'uso di una funzione armonica questa non è altro che la funzione utilizzata in Pinocchio (Baran, et al., 2007). Nella generalizzazione (Hétroy, et al., 2009) questa è veloce quanto il metodo da loro implementato, ma non permette un controllo accurato delle dimensioni della zona deformata.

3.5.2 Generalizzazione con la distanza Euclidea

Calcolare la distanza Euclidea come sappiamo è semplice e veloce. Grazie alla segmentazione si supera il secondo problema della distanza Euclidea analizzato nel paragrafo 2.1.2: le aree di maggior deformazione corrispondono ai bordi trovati, poiché per calcolare le distanze ci si muove proprio dai bordi. Tuttavia si ricade nel primo e nel terzo problema già discussi sempre nel paragrafo 2.1.2: si può dare peso ad un bone a cui il vertice non appartiene anatomicamente. Questo accade perché al solito la distanza Euclidea non tiene conto della geometria e quindi raggiunge tutti i vertici vicini indistintamente, espandendosi da ogni punto del bordo non tramite la

superficie, ma piuttosto in linea d'aria. Si ripensi, in un modello in T-pose, alla vicinanza dei vertici di una gamba alle bones dell'altra gamba.

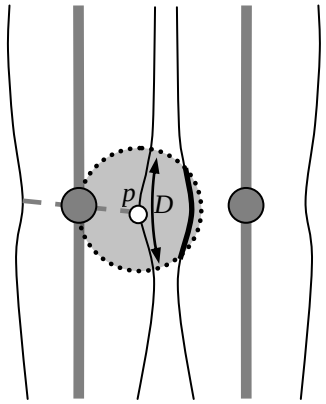


Figura 3.10*: dal confine tratteggiato, in particolare dal punto di confine p appartenente a tale bordo, ci si espande del diametro tramite distanza Euclidea. In questo modo i vertici dell'altro ginocchio, che stanno nella zona marcata all'interno del cerchio punteggiato, vengono influenzati dai bones della gamba opposta, cosa che anatomicamente non dovrebbe accadere.*

Si può pensare di correggere questi problemi sfruttando la gerarchia delle regioni. Tale gerarchia è ereditata da quella dei bones dello skeleton: una regione R_i è figlia di un'altra regione R_j se contiene un bone che ha bone-padre in R_j (ricordiamo come le regioni possono essere viste come dei nuovi bones, o meglio come nuove trasformazioni che sostituiscono il vecchio skeleton). Possiamo quindi impedire ai vertici l'appartenenza ad aree di sovrapposizione di joint che sono lontani dalla loro proto-regione nella gerarchia delle regioni.

Questa correzione è in un certo senso un'approssimazione molto debole della distanza geodetica, in cui la superficie viene appiattita tramite i soli bordi della mesh. Tuttavia questa soluzione non è sempre possibile e comunque può capitare che crei delle zone di discontinuità dove la morbidezza dei pesi sulla superficie cessa di netto.

Un'alternativa che noi abbiamo esaminato era quella di sfruttare la gerarchia, non per escludere vertici dall'area di sovrapposizione, ma per aumentare la distanza tra questi: partendo da un bordo si calcola la distanza da un vertice passando tramite tutti

i bordi intermedi della stessa gerarchia. Anche in questo caso però si nota la presenza di discontinuità.

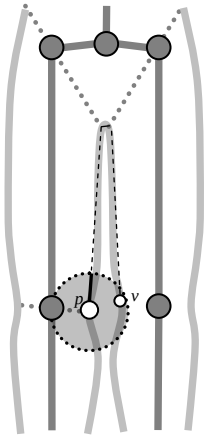


Figura 3.11*: il vertice v sul ginocchio sinistro è dentro la distanza Euclidea presa partendo dal punto di bordo p sul ginocchio destro. Ma se si considera anche la gerarchia delle regioni tramite la linea nera tratteggiata la distanza aumenta tanto da essere troppo distante dal bordo; così il punto di bordo p non può influenzare anche tale vertice v.*

Non bastasse, con la distanza Euclidea si accentuano gli artefatti per il calcolo dei pesi nelle zone più arcuate, come ad esempio sull'inguine.

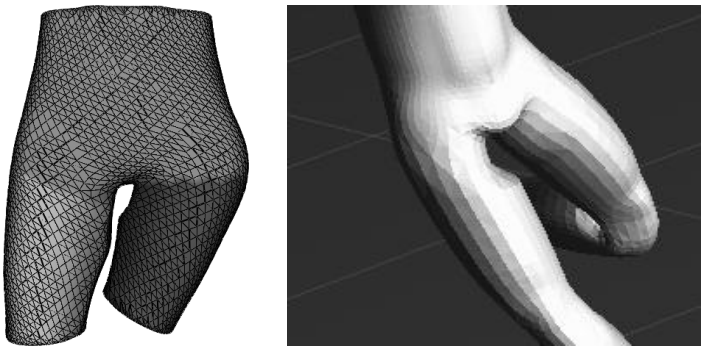


Figura 3.12*: l'uso della distanza Euclidea per il calcolo dei pesi accentua artefatti nelle zone maggiormente arcuate.*

Questo fenomeno si verifica sempre perché la distanza Euclidea non tiene conto della geometria, per cui punti che movendosi sulla superficie hanno una certa distanza da un bordo, in linea d'aria (cioè con la distanza Euclidea) possono essere meno distanti.

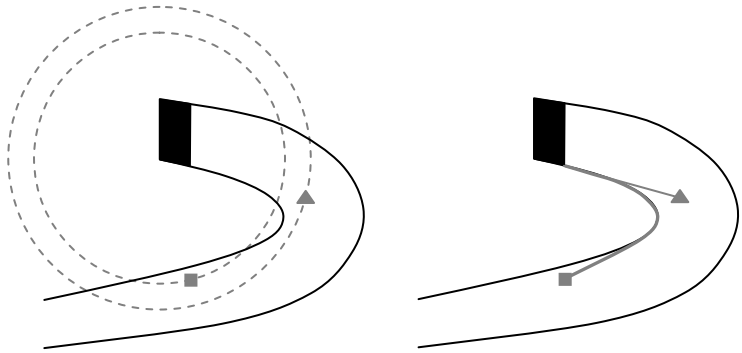


Figura 3.13: *la fascia nera rappresenta un bordo della regione, mentre il quadratino e il triangolino rappresentano due vertici della superficie; con la distanza Euclidea (figura a sinistra) il quadratino è più vicino al bordo rispetto al triangolino, mentre muovendoci sulla superficie (figura a destra) avviene il contrario.*

3.5.3 Secondo metodo di calcolo dei pesi

Nel nuovo metodo cambia anche la funzione di calcolo dei pesi, sfruttando un approccio che non considera più tanto le aree di sovrapposizione, ma i bordi delle regioni estese, le quali sono indicate con R'_j. Ricordiamo che una regione estesa R'_j è formata dall'unione della regione R_j che l'ha generata e dalle aree di estensione che partono da ogni $B_{j,k}$ di R_j verso le regioni esterne a R_j.

Esaminiamo ogni singola regione estesa R'_j. Questa avrà k confini $B'_{j,k}$. Prendiamo la sua mappa di distanza che ad ogni vertice v appartenente ad R'_j dà la sua distanza dal confine della regione stessa. Tale distanza è calcolata come la distanza minima da v a tutti i bordi $B'_{j,k}$, e come abbiamo detto, può essere una distanza Euclidea, o geodetica, o armonica (o qualsiasi altra forma di valutazione di coinvolgimento del vertice da un bordo).

Calcoliamo la distanza massima al confine δ_j dei vertici in R'_j:

$$\delta_j = max_{v \in R'_j} d_j(v)$$

In questo caso $d_j(v)$ è la distanza tra il vertice v e il bordo della regione j, da non confondere con la funzione di distanza $d_{i,j}(v)$ del primo metodo (il quale come

vedremo indica più il peso di due regioni su un vertice che una distanza nel senso classico). Tale distanza come abbiamo detto può essere Euclidea, geodetica, armonica o di altro genere. Prendiamo ora per ogni vertice v dei pesi non normalizzati:

$$\sigma_j(v) = \beta\left(\frac{d_j(v)}{\delta_j}\right)$$

I pesi normalizzati sono quindi definiti come:

$$w_j(v) = \frac{\sigma_j(v)}{\sum_{i \in I(v)} \sigma_i(v)}$$

Dove $I(v)$ è l'insieme di indici delle regioni estese a cui appartiene il vertice v: $I(v) = \{j \in \{1, \dots, n\} : v \in R'_j\}$.

Notiamo che se un vertice *v* appartiene al bordo di una regione R'_j, abbiamo $d_j(v) = 0$, quindi anche $\sigma_j(v) = 0$ e $w_j(v) = 0$.

Un vantaggio che abbiamo da questa formula è che, diversamente dal precedente metodo di calcolo dei pesi, qui non abbiamo bisogno di trattare separatamente i vertici che hanno una sola regione relativa. Infatti per questi vertici $I(v)$ si riduce a un singleton $\{j\}$ e quindi $w_j(v) = 1$. Questo ci viene dal fatto che δ_j è un valore calcolato su tutti i vertici della regione estesa e non solo su quelli delle aree di sovrapposizione. Ciò significa che ogni vertice, avendo almeno una regione, avrà sicuramente almeno anche un $\sigma_j(v)$.

Poiché le regioni estese R'_j definiscono una copertura della superficie e β è monotona da [0,1] su [0,1], il denominatore non sarà mai uguale a zero e si avrà $w_j(v) \in$ [0,1]. Infine, grazie alla normalizzazione finale, per ogni vertice la somma di tutti i pesi dà 1: $\sum_{j \in I(v)} w_j(v) = 1$.

3.6 Analisi

Come accennato nello sviluppo di questo metodo non ci siamo limitati ad una reimplementazione, ma abbiamo tentato un'analisi mirata alla comprensione del metodo (e della sua generalizzazione) e alla ricerca di possibili evoluzioni. Anche se questo algoritmo ha la possibilità di sfruttare informazioni semantiche dello skeleton, in questo studio noi tralasciamo quest'aspetto, che può essere comunque ripreso ed integrato in seguito e ci concentriamo sui singoli passi.

3.6.1 Analisi della decomposizione

Abbiamo detto che la prima fase si occupa di suddividere l'area in regioni. Per fare ciò utilizza dei piani di taglio che tentano di creare dei bordi di lunghezza minima. Esaminiamo cosa accade quando si genera un piano tra due bones b_i e b_j. In particolare supponiamo per ora che il piano sia esattamente il *piano di bisezione* (o *piano bisettrice*) tra i due bones. Tale piano è il luogo dei punti equidistanti dai due bones b_i e b_j, dove in questo caso la distanza punto-bone è intesa come la minima distanza Euclidea (non a caso vedremo). Ricordiamo che i due bones hanno un joint in comune; questa era una condizione imposta per generare i piani, quindi anche il joint in comune fa parte di tale piano. Inoltre il piano di bisezione separa lo spazio in due regioni: R_i che è quella dei punti "più vicini" al bone b_i e R_j che è quella dei punti più vicini all'altro bone b_j.

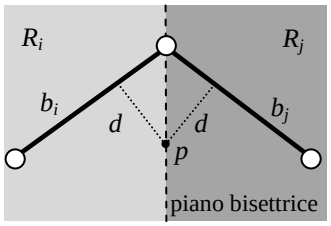


Figura 3.14: *il piano bisettrice indicato dalla linea tratteggiata è il luogo dei punti in cui la minima distanza Euclidea dai due bones b_1 e b_2 è uguale. Le regioni R_1 (in grigio chiaro) ed R_2 (in grigio scuro) sono separate da tale piano di bisezione e rappresentano il luogo dei punti più vicini rispettivamente a b_1 e a b_2. Preso un punto p sul piano bisettrice, la distanza tra p e b_1 e la minima distanza Euclidea tra p e b_2 è la stessa, cioè d.*

Da tutto ciò è intuibile come utilizzare il piano bisettrice equivalga a spartire i vertici tra le due regioni usando la distanza Euclidea: il vertice appartiene ad una regione R_{i1} se il bone ad esso più vicino appartiene a tale regione R_{i1}. E il discorso non vale solo per il partizionamento, ma anche per le distanze stesse: date le 2 distanze tra un vertice e i due bones, più queste sono uguali più il vertice è vicino al piano di bisezione. La distanza dal piano bisettrice in altre parole rappresenta lo "sbilanciamento" del vertice verso una delle due regioni, così come fanno le distanze dai due bones. Questo ci tornerà utile poco più sotto per comprendere il criterio per cui nel paragrafo 3.2.1 venivano generati diversi piani P_1, P_2,…, P_k per un singolo joint. Riassumendo per ora possiamo dire che il piano bisettrice per due bones è un'esplicitazione del criterio della distanza Euclidea (in realtà è "quasi" un'esplicitazione; una vera esplicitazione è descritta di seguito).

Il piano bisettrice però sfrutta la topologia dello skeleton e non della mesh: è come se si espandessero i bones aumentandone il volume, fino a riempire l'intera scena. Questo è lo stesso difetto di fondo di quando si utilizza la distanza Euclidea che ne genera i problemi visti in 2.1.2 . In effetti definire la regione di un bone tramite la distanza Euclidea equivale a sfruttare i piani di un diagramma di Voronoi su segmenti invece che su punti. I diagrammi di Voronoi sono un meccanismo molto potente per capire la struttura di una geometria, quindi vi faremo più volte riferimento. Un diagramma di Voronoi rappresenta infatti una decomposizione di uno spazio tramite distanze rispetto ad un insieme discreto di elementi. Questi elementi possono essere punti, segmenti o altro. Nel nostro caso sono segmenti, ovvero i nostri bones. I piani che delimitano i confini di un diagramma di Voronoi rappresentano quindi le zone equidistanti tra due o più elementi (segmenti), mentre un'area del diagramma rappresenta l'area dei punti più vicini all'elemento (segmento) contenuto nell'area stessa. Dunque i diagrammi di Voronoi sono un'esplicitazione di una decomposizione dello spazio, mentre la distanza Euclidea esprime la stessa decomposizione implicitamente.

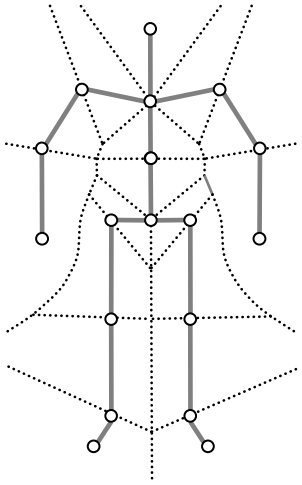


Figura 3.15: *diagramma di Voronoi costruito a partire dallo skeleton. Ogni area è separata da un confine indicato dalle linee punteggiate.*

Per quanto riguarda i piani di bisezione questi non sono altro che alcuni dei piani separatori del diagramma di Voronoi: solo alcuni di quelli passanti per i joints. Mentre i piani di bisezione su cui si basa l'algoritmo che stiamo esaminando sono ancora un sottoinsieme di questi ultimi, ovvero sono solo quelli passanti per i joints con solo 2 bones incidenti.

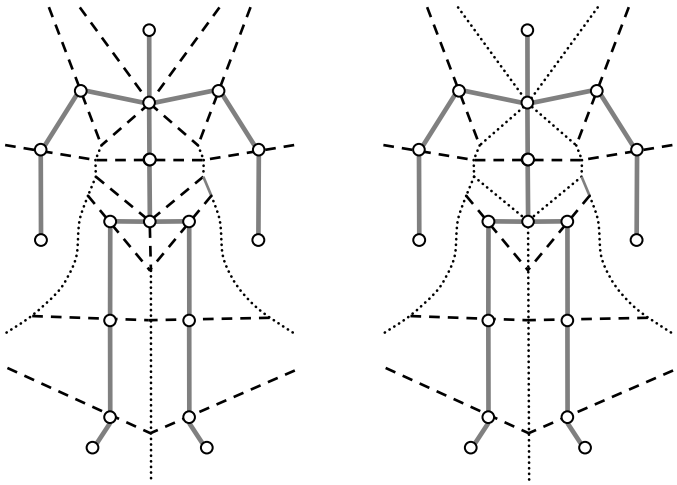


Figura 3.16: *nell'immagine a sinistra sono evidenziati tramite linee tratteggiate i piani di bisezione nel diagramma di Voronoi; nell'immagine a destra sono evidenziati tramite linee tratteggiate i soli piani considerati dalla decomposizione dell'algoritmo basato su Atlas.*

Questi ultimi infatti sono più facilmente definibili tramite un unico piano: si pensi ad esempio al joints centrale sul petto nel quale incidono 4 bones e che quindi avrà 4 piani di bisezione; mentre su un gomito vi è un unico piano separatore passante per un joint.

Come abbiamo detto però in questo modo consideriamo l'anatomia dello skeleton e non della mesh. Per questo l'algoritmo non prende direttamente i piani bisettrice. Si ripensi al secondo problema dovuto ai metodi basati sul criterio della distanza Euclidea: tale criterio, come abbiamo visto, individua come aree di massima deformazione zone che possono essere non anatomicamente realistiche. Questo è dovuto proprio al fatto che con la distanza Euclidea le aree di maggior flessione coincidono esattamente con le zone in cui passa il piano di bisezione.

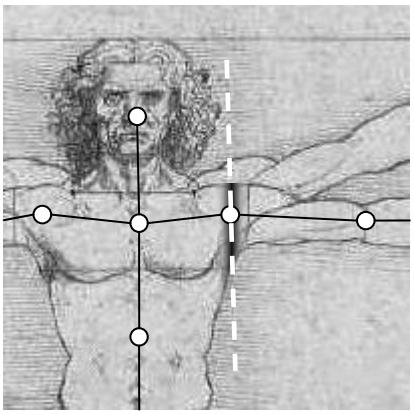


Figura 3.17: *il piano di bisezione non sempre individua le aree di maggior deformazione anatomicamente migliori. In figura il piano di bisezione descritto dalla linea tratteggiata individua come area di maggior deformazione l'area sfumata.*

Quindi, invece di prendere direttamente il piano bisettrice, l'algoritmo tenta di "aggiustare" il risultato valutando diversi piani attorno al joint. Ciò può anche essere visto come una modifica del diagramma di Voronoi sui piani che gli interessano e in base alla mesh, non più allo skeleton. Tra i piani generati viene scelto il piano che genera il confine localmente più piccolo, il quale è un buon candidato come fascia di massima flessione. Si riesce così ad individuare un bordo più adatto anche quando una regione anatomica si espande in modo maggiore della sua adiacente come nel caso già

esaminato della spalla: il torace è molto più largo del braccio e quindi il piano bisettrice passante per la spalla può non tagliare nella parte più adatta dell'ascella.

Quest'ultimo esempio ci fa intuire qualcosa in più sulle cause di questo fenomeno. Da un punto di vista puramente anatomico, questo problema deriva dalla struttura dello skeleton, il quale è una estrema semplificazione di uno scheletro umano di cui dovrebbe simulare il comportamento. Il problema è che lo scheletro umano è costituito da un numero sicuramente maggiore di ossa e queste sono di forma e struttura molto più complessa di quelle che usiamo noi nel nostro skeleton. Nello specifico l'uso degli skeleton con i quali di solito lavoriamo comporta in qualche modo una "perdita di volume" rispetto all'uso di uno scheletro simil-reale. Per esempio, rispetto all'area dell'ascella, la profondità dalla superficie non dovrebbe coinvolgere il bone della clavicola, ma un osso della scapola e/o di una costola. Tuttavia a noi torna comodo utilizzare skeleton semplificati, sia per gestire i dati, sia per facilitare l'approccio, sia per ottimizzare le performance.

Il partizionamento tramite piani di taglio che si adattano all'anatomia del modello pone in qualche modo rimedio a questo difetto. Notiamo anche che le normali $\{n_1, n_2,\ldots, n_k\}$ scelte, giacciono tutte sullo stesso piano Q su cui giacciono anche i due bones. Questo, assieme all'imposizione che i piani passino per J, garantisce che venga considerata in una certa misura anche l'anatomia dello skeleton.

3.6.2 Problematiche della decomposizione tramite singoli piani

Il fatto che i piani scelti individuino esplicitamente e con una buona approssimazione le fasce di massima deformazione dell'anatomia è uno dei punti di maggior forza di questo metodo. Tuttavia l'approccio basato su piani di taglio può presentare in generale diversi difetti. Anzitutto suppone che i piani non si intersechino mai all'interno della mesh. Questo avviene facilmente per le mesh testate dagli autori dell'algoritmo Atlas-based (Lu, et al., 2008), in quanto modelli stelliformi con joints

posizionate in modo ottimale al partizionamento. Anche per questo motivo ogni bordo generato risulta poi essere il confine tra 2 sole regioni. Questa però non è una proprietà garantita per mesh generiche. Si pensi ad esempio ad una mesh di un personaggio con una gonna (anche non troppo lunga) o con un cavallo dei pantaloni molto basso. In questi casi può essere difficile e a volte impossibile trovare un unico piano di taglio sulle anche che non dia fastidio ad altri piani generati dalla decomposizione.

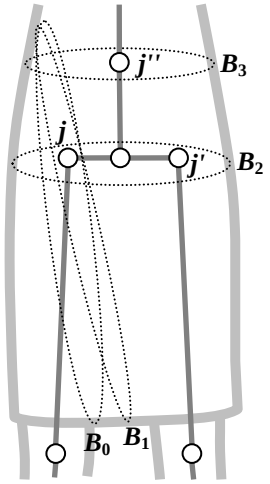


Figura 3.18: *tra i piani generati per il joint j consideriamo quelli che generano i bordi B_0, B_1 e B_2. Il bordo B_2 è quello di lunghezza più piccola tra tutti, quindi sarebbe quello scelto dall'algoritmo; tuttavia tale bordo coincide con quello generato per j' e questo causerebbe problemi per il partizionamento. Si potrebbe pensare che il bordo più adatto sia allora B_0 o B_1, perché separano la coscia dal bacino, tuttavia anche tali piani causerebbero problemi intersecando il piano B_3 scelto per il joint j'' dell'addome. La realtà è che il partizionamento più adatto non passa per un singolo piano di taglio.*

La cosa risulta evidente anche dal discorso fatto sui diagrammi di Voronoi: i piani di separazione sono definiti da un singolo piano solo in condizioni ottimali. Un'idea per superare il problema sarebbe spostare i joint, ma questo porterebbe ad una valutazione dell'anatomia scorretta. Per esempio abbassando i joint del bacino si accorcerebbero le cosce, sconvolgendo la corretta anatomia.

In secondo luogo abbiamo detto che i confini devono corrispondere alle aree in cui la mesh si deforma maggiormente. Ma tali confini non sempre sono definibili in maniera ottimale tramite un unico piano. Si prenda ancora l'esempio della spalla. Al

di sotto di essa il punto di deformazione maggiore corrisponde al punto più interno dell'ascella. Al di sopra della spalla però il punto di maggiore deformazione in generale non è simmetrico al punto al di sotto rispetto al joint. Tale problema si genera dal fatto che il partizionamento avviene sullo skeleton e viene aggiustato sulla geometria della mesh solo in modo approssimativo tramite un singolo piano.

Ultimo difetto evidente del metodo (voluto dagli autori per motivi di semplificazione) è che si segmenta la mesh solo su joints che connettono 2 bones. In altre parole si riduce il numero di bones dello skeleton: più di 2 bones (che ricordiamo rappresentano delle trasformazioni) connesse da un unico joint sono fuse in un'unica trasformazione rigida. Ecco allora che una regione equivale ad una singola trasformazione generata a partire da più bones. Per questo i pesi sono definiti sulle regioni e non sui bones. Questo fa parte dell'approccio scelto dagli autori per semplificare al massimo l'individuazione dei piani di taglio. Ovviamente tale scelta limita alcune particolari pose nell'animazione. Per quanto riguarda modelli umani potrebbe essere abbastanza ininfluente per certe parti, come ad esempio i bones che costituiscono il bacino, dato che questi hanno tra loro una rigidità naturale, ma per altre parti comporta la perdita di alcuni movimenti, ad esempio le rotazioni delle due clavicole, la spina dorsale e la base del collo.

3.6.3 Analisi della funzione β

Per analizzare i 2 metodi di calcolo dei pesi presentati, cominciamo con lo studiare la funzione β che viene utilizzata sia nel primo che nel secondo metodo. Osserviamo intanto come si comporta tale funzione. Tanto per mettere in evidenza quali sono le radici della funzione possiamo riscrivere la funzione come:

$$\beta(l) = l^2(-2l + 3)$$

Dunque le radici sono 0 e 2/3 (cioè 1,5). Questa è ovviamente una funzione definita sui reali. Tuttavia, nella nostra formula per il calcolo dei pesi, passiamo a β solo valori restituiti dalla funzione $d_{i,j}(v)$ nel primo metodo e dalla funzione $\sigma_j(v)$ nel secondo. Entrambe queste funzioni restituiscono solo valori tra 0 e 1. Quindi per la funzione β ci interessano solo i valori restituiti passando come argomenti valori reali tra 0 e 1.

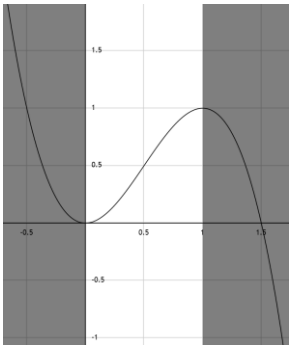


Figura 3.19: *la funzione β riceve solo valori restituiti da $d_{i,j}$ o da $\sigma_j(v)$, quindi solo valori tra* 0 *e* 1.

Notiamo anzitutto che a valori del dominio in [0,1] corrispondo valori del codominio [0,1]. Dunque se (come in effetti facciamo) passiamo a β solamente valori tra 0 e 1, allora β stesso ci darà solo valori tra 0 e 1. Questo è bene, perché usiamo β proprio per calcolare dei pesi, che sono valori tra 0 e 1. Osserviamo anche come questa funzione si curvi dolcemente in prossimità di 0 e 1, mentre diventa più lineare avvicinandosi a 0,5, punto dove si bilancia perfettamente. Sfruttando quindi β per trattare i valori restituiti da $d_{i,j}(v)$ o da $\sigma_j(v)$, li rendiamo più dolci in prossimità dei margini in cui i vertici cominciano a diventare rigidi.[16]

[16] Si può ritrovare la funzione β in una parte della spline cubica di Hermite, in particolare nell'interpolazione sull'intervallo [0,1] (e non è un caso): β è infatti una delle 4 funzioni base di Hermite tramite cui si definisce il polinomio $p(t) = h_{00}(t)p_0 + h_{10}(t)m_0 + h_{01}(t)p_1 + h_1(t)m_1$

- $h_{00}(t) = 2t^3 - 3t^2 + 1 = (1 + 2t)(1 - t^2)$
- $h_{10}(t) = t^3 - 2t^2 + t = t(1 - t)^2$
- $\boldsymbol{h_{01}(t) = -2t^3 + 3t^2 = t^2(3 - 2t)}$
- $h_{11}(t) = t^3 - t^2 = t^2(t - 1)$

3.6.4 Analisi del calcolo dei pesi del primo metodo

Analizziamo ora il primo metodo per calcolare i pesi. Guardiamo quindi la funzione $d_{i,j}(v)$, composta da 3 casi, utilizzata per calcolare i pesi e tentiamo di capire come opera secondo l'idea degli autori.

$$d_{i,j}(v) = \begin{cases} \dfrac{\frac{D_{i,j}}{2} + g(v, B_{i,j})}{D_{i,j}} & \exists B_{i,j} \ \& \ R_i \text{ è } proto-regione \text{ di } v \\ \dfrac{\frac{D_{i,j}}{2} - g(v, B_{i,j})}{D_{i,j}} & \exists B_{i,j} \ \& \ R_j \text{ è } proto-regione \text{ di } v \\ 0.5 & B_{i,j} \text{ non esiste} \end{cases}$$

$$i = 1,2, \dots n; \ j = 1,2, \dots n; \ j \neq i$$

In particolare studiamo i primi due casi in cui esiste il bordo. Poiché $D_{i,j}$ è il diametro del bordo $B_{i,j}$, banalmente $\frac{D_{i,j}}{2}$ rappresenta il raggio del bordo. La formula viene applicata solo per vertici che si trovano entro il raggio partendo dal bordo, cioè nelle aree di sovrapposizione, dunque sicuramente la distanza di tali vertici dal bordo $g(v, B_{i,j})$ è minore del raggio (e maggiore di 0 per la stessa definizione di distanza geodetica): $0 \leq g(v, B_{i,j}) \leq \frac{D_{i,j}}{2}$. Ciò significa che più $g(v, B_{i,j})$ è uguale al valore di $\frac{D_{i,j}}{2}$ (cioè al raggio), più la loro somma (primo caso) diventa uguale al diametro e viceversa la loro differenza (secondo caso) diventa uguale a zero. Prendiamo per esempio i casi estremi in cui $g(v, B_{i,j}) = \frac{D_{i,j}}{2}$ e $g(v, B_{i,j}) = 0$. Per $g(v, B_{i,j}) = \frac{D_{i,j}}{2}$ avremo:

- $\frac{\frac{D_{i,j}}{2} + g(v, B_{i,j})}{D_{i,j}} = \frac{\frac{D_{i,j}}{2} + \frac{D_{i,j}}{2}}{D_{i,j}} = \frac{D_{i,j}}{D_{i,j}} = 1$ nel primo caso, e
- $\frac{\frac{D_{i,j}}{2} - g(v, B_{i,j})}{D_{i,j}} = \frac{\frac{D_{i,j}}{2} - \frac{D_{i,j}}{2}}{D_{i,j}} = \frac{0}{D_{i,j}} = 0$ nel secondo caso.

Mentre per $g(v, B_{i,j})$ = 0 avremo:

$\frac{\frac{D_{i,j}}{2} \pm g(v,B_{i,j})}{D_{i,j}} = \frac{\frac{D_{i,j}}{2} \pm 0}{D_{i,j}} = \frac{\frac{D_{i,j}}{2}}{D_{i,j}} = \frac{1}{2}$ sia nel primo che nel secondo caso.

Ciò significa che abbiamo:

$0.5 \leq d_{i,j}(v) \leq 1$ nel primo caso e $0 \leq d_{i,j}(v) \leq 0.5$ nel secondo caso.

La cosa diventa ancora più evidente se semplifichiamo i due casi:

$$\frac{\frac{D_{i,j}}{2} \pm g(v, B_{i,j})}{D_{i,j}} = \frac{\frac{D_{i,j} \pm 2g(v, B_{i,j})}{2}}{D_{i,j}} = \frac{D_{i,j} \pm 2g(v, B_{i,j})}{2D_{i,j}} = \frac{\cancel{D_{i,j}}}{2\cancel{D_{i,j}}} \pm \frac{\cancel{2}g(v, B_{i,j})}{\cancel{2}D_{i,j}}$$
$$= \frac{1}{2} \pm \frac{g(v, B_{i,j})}{D_{i,j}}$$

Dunque la funzione è semplificabile come:

$$d_{i,j}(v) = \begin{cases} 0.5 + \frac{g(v, B_{i,j})}{D_{i,j}} & \exists B_{i,j} \text{ \& } R_i \text{ è } proto-regione \text{ di } v \\ 0.5 - \frac{g(v, B_{i,j})}{D_{i,j}} & \exists B_{i,j} \text{ \& } R_j \text{ è } proto-regione \text{ di } v \\ 0.5 & B_{i,j} \text{ non esiste} \end{cases}$$

$$i = 1,2, \dots n;\ j = 1,2, \dots n;\ j \neq i$$

In altre parole la funzione, nel caso in cui esista un bordo, cioè nei primi due casi, restituisce un valore che si sposta da 0.5 incrementandolo o decrementandolo a seconda di quale è la proto-regione del vertice.

Il terzo caso è, in un certo senso, un caso intermedio, per cui viene fatto restituire un valore bilanciato per garantire una sorta di continuità nella definizione di $d_{i,j}(v)$.

Più che una funzione di distanza nel senso classico del termine, potremmo dire quindi che questa è una funzione che misura il "grado di coinvolgimento" del vertice v rispetto a due regioni: $d_{i,j}(v)$ ci dice quanto il vertice v è influenzato rispetto alle regioni i e j. In altri termini $d_{i,j}(v)$ indica già di per sé come il peso è distribuito rispetto alle sole due regioni i e j; ma lo fa in modo lineare, per questo trattiamo i pesi che restituisce tramite β che li ammorbidisce laddove stanno per diventare rigidi.

3.6.5 Analisi del calcolo dei pesi del secondo metodo

Abbiamo detto che questo metodo sfrutta il rapporto tra la distanza (stavolta intesa proprio nel suo senso classico) d_j e la massima distanza δ_j per calcolare un peso che indica quanto v è coinvolto dalla regione estesa R_j. Anche questo peso è lineare e quindi anche qui lo trattiamo tramite β ottenendo il peso σ_j. Dunque in questa formalizzazione il grado di coinvolgimento (cioè il peso) è definito da σ_j e ben separato dal concetto di distanza, espresso invece da d_j. Stavolta comunque i pesi non sono normalizzati e quindi li normalizziamo al momento del calcolo del peso w_j.

La chiave per comprendere il principio di quest'altro metodo è il valore δ_j. Tale valore rappresenta in qualche modo la distanza del vertice più interno di tutti quelli appartenenti alla regione estesa considerata. Quindi δ_j dà una misura approssimata della massima profondità a cui ci si può spingere all'interno della regione. Diversamente dal metodo precedente, il punto di riferimento questa volta non sono più i bordi della regione, dai quali ci si muoveva per aumentare o diminuire il peso. In questo caso il punto di riferimento diventa la parte più interna alla regione estesa e i bordi di quest'ultima. In altre parole δ_j indica la distanza per raggiungere il vertice "più rigido" di tutta la regione estesa, quello più lontano da tutti i bordi e quindi meno influenzato da questi. Trovato tale valore, ricalcando un po' l'intuizione di Talete con

l'ombra della piramide di Cheope[17], lo si utilizza come termine di paragone comune per tutte le distanze di tutti gli altri vertici, per stabilire quanto ogni vertice si allontana dalla parte rigida avvicinandosi al bordo.

In questo modo il blending dei pesi non si svolge più solo sui vertici interni alle aree di sovrapposizione, ma su tutti i vertici. Tuttavia, per i vertici al di fuori di ogni area di sovrapposizione, la rigidità si ristabilisce nel momento del calcolo effettivo del peso; questo perché, come abbiamo visto, la sommatoria al denominatore si riduce ad un singleton ed in particolare si riduce ad un σ uguale al numeratore, cosicché la frazione ritorna 1 sull'unica regione coinvolta:

$$w_j(v) = \frac{\sigma_j(v)}{\sum_{i=j} \sigma_i(v)} = \frac{\sigma_j(v)}{\sigma_j(v)} = 1$$

Comunque, mentre nel primo metodo ci bastava avere la distanza geodetica pre-calcolata al passo precedente dai bordi della regione, in questo secondo metodo è necessario un ricalcolo delle distanze, stavolta dai bordi della regione estesa.

3.7 Nuova decomposizione della mesh

Dall'analisi precedente, come prima idea per migliorare la decomposizione, si potrebbe pensare di usare più di un piano per ogni joints. Ad esempio per le joints su cui incidono esattamente solo 2 bones si potrebbe generare un semi-piano per la parte di angolo convesso formato dai due bones ed un altro semi-piano per la parte di angolo concavo. In tal modo si individuerebbe una fascia di massima deformazione

[17] Talete di Mileto fu un antico filosofo e matematico greco. Come racconta Plutarco, durante un suo viaggio in Egitto fu sfidato dal faraone Amasis a misurare l'altezza della piramide di Cheope. Talete vi riuscì con un semplice bastone, calcolando il rapporto tra l'ombra della piramide e quella del suo corpo, sfruttando l'ora del giorno in cui la nostra ombra ha la stessa lunghezza della nostra altezza. Come Talete usò il sole e quindi le ombre come paragone universale per le altezze così noi sfruttiamo la massima profondità δ_j come paragone universale per la rigidità.

anatomicamente più precisa. Oppure si potrebbero anche generare semi-piani per i joints che hanno più di 2 bones incidenti, così da non dover rinunciare ad alcuni movimenti su tali bones.

Tuttavia, sempre per i motivi citati nell'analisi, abbiamo deciso di cambiare totalmente metodo di partizionamento, poiché questo è causa dei difetti e dei pregi (in particolare l'individuazione delle zone di maggior deformazione) del metodo. Nel fare ciò abbiamo cercato un metodo che tenesse conto il più possibile della geometria e dell'anatomia. Pensando a questo abbiamo preso in considerazione i metodi che generano skeleton: questi metodi infatti partono da una mesh ed ottengono uno skeleton. Partire dalla mesh significa partire dai suoi vertici, ovvero dalla geometria. Dunque se nel procedimento di creazione dello skeleton ci portiamo dietro i vertici che vengono usati per "costruire" un bone abbiamo un legame naturale e diretto tra vertici e bones che tiene fortemente conto della geometria.

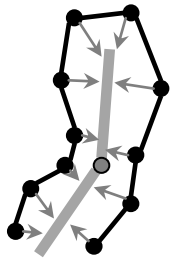


Figura 3.20: *i metodi che generano skeleton partendo dai vertici di una mesh generano i bones.*

Per avere più chiara la cosa si ripensi alla generazione di un diagramma di Voronoi, ma stavolta partendo dalle facce della mesh e non dai segmenti dello skeleton. Se tra i piani di suddivisione dello spazio del diagramma prendiamo quelli interni alla mesh, osserviamo che definiscono una struttura della mesh stessa. Da tale struttura si può estrarre uno superficie scheletrica o anche uno skeleton. Difatti questo è uno dei più comuni metodi utilizzati per generare uno skeleton. Inoltre partendo dalle facce per generare i piani, se ci portiamo dietro i loro vertici, per ogni piano sappiamo da quali vertici ha avuto origine e così anche per un eventuale skeleton estratto.

Utilizzare un algoritmo che genera skeleton ci garantirebbe quindi una buona correttezza anatomica del partizionamento. Tuttavia noi non vorremmo generare uno skeleton da zero, ma usarne uno esistente. Fortunatamente diversi metodi di rigging, i quali si occupano di inserire uno skeleton esistente nella mesh, seguono gli stessi principi della generazione di skeleton e quindi ne ereditano gli aspetti e i procedimenti che desideriamo. Decidiamo quindi di utilizzare un metodo di rigging dello skeleton (o almeno parte dei calcoli che sarebbero necessari per il rigging) per ottenere un partizionamento.

Teniamo comunque anche da conto che se un giorno si volesse estrarre pure uno skeleton, magari per altre forme anatomiche diverse, i principi base di questo metodo non solo varrebbero ancora, ma sarebbero anche riutilizzabili per l'estrazione stessa.

Sottolineiamo anche il fatto che con questo metodo, diversamente dal metodo di segmentazione originale basato su piani di taglio unici, si genererà una partizione per ogni singolo bone: dunque avremo tante regioni quanti sono i bones dello skeleton. Se volessimo comunque ricondurci all'algoritmo di Atlas-based originale (Lu, et al., 2008), una volta calcolata una regione per ogni bone, potremmo fondere le regioni che contengono bones incidenti in joints di cardinalità maggiore a 2. Tuttavia abbiamo deciso di evitare di tornare a tale partizionamento, poiché come abbiamo visto limita alcuni movimenti. Dunque nella nostra versione dell'algoritmo più di due bones connessi da uno stesso joint non saranno più considerate rigide tra loro e riunite in un'unica trasformazione.

Ribadiamo ancora una volta che, con questo metodo di segmentazione, non ci muoviamo più dallo skeleton alla superficie tentando poi di aggiustare i risultati in base alla geometria, come avveniva nell'algoritmo originale, ma viceversa ci muoviamo dalla superficie (e quindi dalla geometria e dall'anatomia) verso lo skeleton.

3.7.1 Nuovo metodo di segmentazione

Come primo tentativo abbiamo provato un metodo di *thinning* (assottigliamento) basato su voxellizzazione. L'intera scena viene suddivisa in voxel (cubi di dimensioni uguali). Si immerge poi lo skeleton nella scena voxellizzata ottenendo così dei voxel-skeleton o voxel-bones. Si inseriscono quindi i vertici della mesh nei voxel. Da questi ultimi poi ci si spinge verso i voxel adiacenti più interni fino a raggiungere i voxel-bones. Nello spingersi in profondità ci portiamo dietro i vertici di partenza che quindi verranno legati al bone del voxel-bone raggiunto.

Ovviamente la velocità del metodo dipende dal numero di voxel generati, ossia dall'unità di grandezza scelta per il voxel. Usando un numero maggiore di voxel il risultato migliora, ma il tempo di calcolo aumenta. Con un numero minore di voxel invece il metodo è più veloce, ma di contro il partizionamento segue molto l'andamento di piani verticali, orizzontali. Questo perché lo spostamento verso voxel adiacenti segue appunto traiettorie orizzontali e verticali. Altro problema è che se il voxel non è abbastanza piccolo può contenere vertici che appartengono ad arti diversi. Si pensi ad esempio ai vertici sulle cosce in un modello in T-pose. In tal caso si dovrebbe intervenire nel primo passo dell'algoritmo separando i vertici e spingendoli nelle rispettive direzioni corrette.

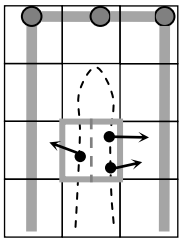


Figura 3.21: *i vertici del voxel vanno separati e spinti nelle rispettive direzioni.*

Per queste problematiche abbiamo deciso di testare un'altro metodo di rigging già implementato nella libreria open source Pinocchio (Baran, et al., 2007). Tale libreria nel corso del rigging genera una certa quantità di sfere interne alla mesh che possono essere viste come delle bounding-spheres.

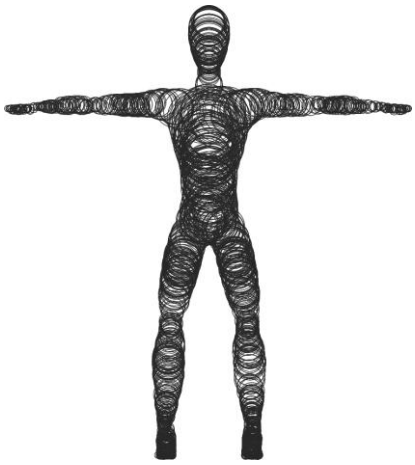


Figura 3.22: *la mesh viene completamente riempita di sfere.*

Le bounding-geometry in generale sono forme geometriche molto semplici (sfere, cubi, ecc.) che vengono composte assieme in modo da approssimare un volume più complesso. Svolgendo i conti su tali geometrie si semplificano e velocizzano calcoli che altrimenti possono essere molto complessi. Le bounding-spheres utilizzano appunto delle sfere per approssimare una geometria complessa. Per effettuare il partizionamento troviamo dei percorsi che attraversano le sfere da ogni vertice sino al bone più vicino. Questo è un approccio simile al precedente basato su voxel, ma abbiamo alcuni vantaggi. Muovendoci rimanendo sempre all'interno di sfere che si intersecano si garantisce di rimanere sempre all'interno della mesh, mentre con i voxel, come abbiamo visto, questo dipendeva dalla dimensione scelta per il singolo voxel. Inoltre le sfere non hanno dimensione fissa come i voxel, quindi muovendoci da una sfera verso quelle che la intersecano possiamo prendere delle direzioni che non sono per forza ortogonali. Oltretutto attraversando sfere molto grandi si effettuano passi di lunghezza maggiore, velocizzando il metodo.

Come noto le bounding-spheres sono poco adatte ad approssimare figure spigolose, ma nel nostro caso le figure umanoidi tendono ad essere abbastanza tondeggianti e quindi per queste si sono dimostrate un buono strumento.

Oltre a questo Pinocchio “involontariamente” fornisce alcune strutture su queste sfere che ci permettono di calcolare i percorsi in tempi estremamente bassi. Tali strutture in realtà sono necessarie al rigging, come avevamo accennato e quindi il tempo necessario alla loro costruzione è incluso in questo. Analizzeremo nel dettaglio tali strutture ed il loro utilizzo più avanti parlando dell’implementazione.

3.7.2 Nuovo rilevamento del confine

Alla fine del procedimento precedente avremmo quindi i vertici suddivisi in tante regioni quante sono i bones. Ora dobbiamo trovare i confini tra le regioni. Possiamo trovare facilmente gli edge di confine: questi saranno semplicemente gli edge che hanno i 2 vertici estremi in regioni diverse. Come nel caso procedente (ma non con lo stesso procedimento), da ogni edge di confine calcoliamo un punto di confine. Ricordiamo le osservazioni fatte sui piani di divisione tra due regioni. Informalmente possiamo dire che un punto di confine sarà un punto che giace sull’edge di confine tra due regioni e che ha “distanza dalle due regioni più equilibrata possibile”. Poiché nella nostra versione dell’algoritmo ad ogni regione corrisponde esattamente un bone, possiamo prendere come distanza tra le 2 regioni la distanza dai 2 relativi bones. Per spiegarci possiamo vedere la cosa nel seguente modo: dato un punto sull’edge abbiamo le 2 distanze tra il punto e i 2 bones delle regioni; prendiamo quindi un punto sull’edge e lo muoviamo rimanendo sull’edge stesso, ma spostandoci in modo da minimizzare la differenza tra le due distanze.

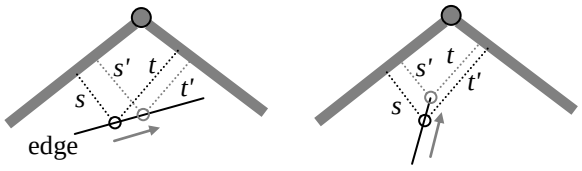


Figura 3.23: *muovendoci sull’edge di confine ci si sposta fino a rendere il segmento s il più possibile di lunghezza uguale al segmento t, ottenendo così s' e t'. Nell’immagine a sinistra si riesce a rendere s' lungo esattamente quanto t', mentre nell’immagine a destra questo non è possibile perché si uscirebbe dall’edge, quindi ci si ferma su un’estremità dell’edge.*

Facciamo un'osservazione. Ipotizziamo che i due bones delle regioni siano connessi (cosa che nel nostro metodo può capitare e sperimentalmente si è verificata spesso, ma non è sempre garantita: possono capitare confini anche tra 2 bones che non hanno joint in comune) e che quindi abbiano un joint in comune. Ipotizziamo anche che quando muoviamo il punto lungo l'edge di confine riusciamo a ridurre la differenza tra le due distanze a 0 (cioè la distanza tra il punto ed uno dei due bones è esattamente uguale alla distanza tra il punto e l'altro bone) rimanendo sempre sull'edge. In questo caso particolare il punto di confine trovato è esattamente pari a quello che troverebbe l'algoritmo Atlas-based nel caso in cui il piano passante per joint comune sia quello di bisezione. Rimanendo all'interno dell'edge di confine si garantisce di rispettare la geometria valutata tramite il partizionamento, mentre bilanciando sulla distanza dei due bones si spinge il punto verso la zona di massima deformazione tra i due bones stessi (che, se i due bones sono connessi, tale zona altri non è che il joint che hanno in comune).

3.8 Nuova estensione delle regioni

Il calcolo della distanza geodetica può essere pressoché identico all'originale. Vediamo invece quel che concerne il calcolo della distanza di estendibilità.

Nell'algoritmo originale, per ogni boundary, si calcola la massima distanza da cui possiamo spostarci dal boundary stesso per estendere la regione. Abbiamo detto che come massima distanza si prende il diametro approssimato perché aiuta ad evitare artefatti di collapsing elbow. Estendere un boundary del diametro verso le due regioni adiacenti, significa estenderlo di una lunghezza pari al raggio dal lato che dà verso una delle due regioni e poi estenderlo sempre del raggio dal lato opposto che dà verso l'altra regione.

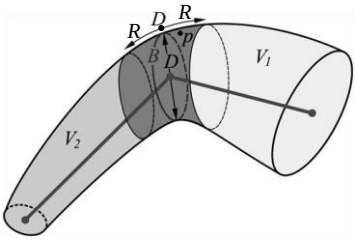


Figura 3.24: *l'estensione centrata nel bordo di una distanza pari al diametro D equivale all'estensione dal bordo di una distanza pari al raggio R verso le due regioni adiacenti al bordo.*

In effetti il raggio *R* è pari proprio a *D* * 0,5 come abbiamo già visto. In realtà, la proprietà di evitare il collapsing elbow viene proprio dall'utilizzo del raggio, in quanto il raggio rappresenta la distanza tra la superficie e lo skeleton.

Nel nostro nuovo algoritmo di decomposizione, poiché i punti di confine di uno stesso boundary possono confinare con regioni diverse, può capitare che il boundary coinvolga più di due bones anche non direttamente connesse da un unico joint. Quindi non abbiamo modo di scegliere univocamente un joint da usare per stabilire il raggio. Si pensi ad uno dei bones del bacino che connette il bone della spina dorsale al bone della gamba. Nel caso in cui la zona dell'inguine sia molto in basso la regione di tale bone può avere anche un unico bordo che da su tutte le regioni confinanti.

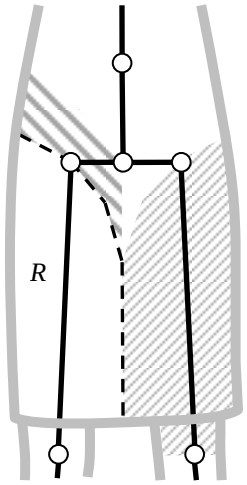


Figura 3.25: *per la regione R il bordo non dà su un'unica regione, ma dà su due regioni diverse indicate tramite tratteggi diversi.*

Dunque non possiamo sfruttare un joint in particolare. Tuttavia avendo un solo bone per regione possiamo calcolare la distanza dalla superficie al bone della regione adiacente per ogni singolo punto bordo. Per un dato bordo quindi prendiamo la media o la minima delle distanze dei punti bordo. Da notare che, diversamente dall'algoritmo originale, nel nostro algoritmo il raggio di estensione può essere diverso nelle 2 direzioni del boundary. Inoltre tale raggio difficilmente sarà simile a quello di "Atlas", perché si basa su un modo completamente diverso di valutare la distanza dalla superficie. Questo approccio ha mostrato comunque risultati soddisfacenti.

3.9 Nuovo calcolo dei pesi

Per il calcolo effettivo dei pesi utilizziamo una specie di ibrido tra le due versioni precedentemente analizzate.

Abbiamo detto che nel secondo metodo δ_j rappresenta in qualche modo un termine di paragone per tutti i vertici della stessa regione estesa, per stimare quanto si è distanti dalla rigidità. Più la distanza $d_j(v)$ è vicina a δ_j più il loro rapporto dà 1 cosicché j ha più peso sul vertice.

Tuttavia utilizzare δ_j come termine di paragone comune può essere esagerato e comportare problemi. Prendiamo per esempio due regioni estese R'_1 ed R'_2 che hanno un'area di sovrapposizione in comune. Supponiamo che R'_1 abbia un'area interna rigida molto ampia, mentre l'area interna rigida di R'_2 sia molto piccola. In questo caso δ_1 sarà molto più grande di δ_2. Questi due valori sono al denominatore del calcolo dei σ. Ciò significa che nel calcolo dei σ_1 della regione R'_1 verranno valori molto più ridotti dei σ_2 di R'_2 e così il peso si sbilancia sulla seconda regione. Questo può comportare che le aree di massima deformazione si discostino dai bordi della

decomposizione, mentre da quanto detto in precedenza, i bordi identificano proprio le aree di massima deformazione anatomicamente più coerenti.

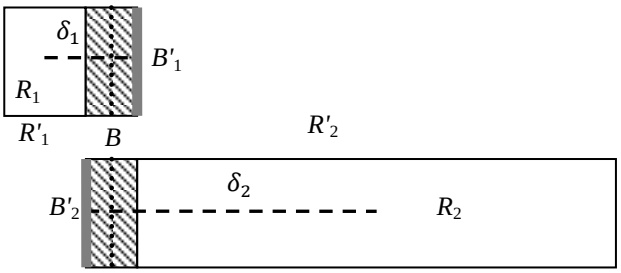


Figura 3.26: *le due regioni estese hanno in comune la stessa area di sovrapposizione segnata da un tratteggio obliquo. Tuttavia i loro δ sono molto diversi. Il bordo che separa R_1 da R_2 è $B_1 = B_2 = B$, bordo presente in entrambi i disegni disegnato con una linea punteggiata. Se prendiamo un punto su tale bordo B questo sarà equidistante da B'_1 e da B'_2 (segnati tramite spesse linee grigie): diciamo che tale distanza è d. Poiché δ_1è molto più piccolo di δ_2, avremo $\sigma_1 = \beta\left(\frac{d}{\delta_1}\right) < \beta\left(\frac{d}{\delta_2}\right) = \sigma_2$. Quindi quando andremo a calcolare w_j il peso si sbilancerà, mentre, poiché il punto è preso sul bordo B, vorremmo che i due pesi fossero bilanciati su tale punto.*

Cerchiamo quindi una stima più adatta di δ. Le distanze $d_j(v)$ sono definite per ogni vertice v e bordo j della regione estesa. Nel calcolare la distanza geodetica per ogni v della regione estesa partendo dai bordi, non solo otteniamo le distanze $d_j(v)$, ma possiamo anche risalire al bordo più vicino $B_{j,k}$ da cui è stata calcolata tale distanza. Come sappiamo ad ogni bordo è associato un diametro $D_{j,k}$ di estensione. Tale diametro indica anche lunghezza dell'area di sovrapposizione o, se preferiamo, indica la distanza tra la parte rigida interna alla regione e il bordo della regione estesa: cioè la distanza tra $B_{j,k}$ e $B'_{j,k}$. Quindi il diametro è una buona stima per sapere, partendo dal bordo $B_{j,k}$, a che distanza i vertici cominciano a diventare rigidi. Inoltre possiamo prendere il diametro più adatto a seconda del bordo più vicino: $D_{j,k}$ è il diametro del bordo $B_{j,k}$ che è lo stesso bordo per cui è stata calcolato $d_j(v)$. I pesi non normalizzati diventano quindi:

$$\sigma_j(v) = \beta\left(\frac{d_j(v)}{D_{j,k}}\right)$$

Tuttavia questo vale per $D_{j,k} > d_j(v)$, ovvero se il vertice v si trova nell'area di estensione di lunghezza $D_{j,k}$ e definita a partire da $B_{j,k}$.

Nel caso il vertice sia fuori dall'area di estensione, cioè nel caso sia un vertice rigido, dobbiamo forzare il peso ad 1 come avveniva nel primo metodo.

$$\sigma_j(v) = \begin{cases} \beta\left(\frac{d_j(v)}{D_{j,k}}\right) & \text{se } D_{j,k} > d_j(v) \\ 1 & \text{altrimenti.} \end{cases}$$

Questo è un po' quello che si faceva nel primo metodo di calcolo del peso, ma invece di partire da 0,5 in riferimento al bordo $B_{j,k}$ e spostarci verso l'interno o verso l'esterno, in questo caso ci muoviamo tra $B'_{j,k}$ e il bordo che delimita la parte rigida della regione, così da definire valori tra 0 e 1.

Come ultima operazione normalizziamo i pesi esattamente come facevamo nel secondo metodo: $w_j(v) = \frac{\sigma_j(v)}{\sum_{i \in I(v)} \sigma_i(v)}$

Tuttavia muovendoci dai bordi per calcolare i σ garantiamo che le aree di massima deformazione coincidano con i bordi stessi.

Capitolo 4

Implementazione

Passiamo ora all'aspetto implementativo. Abbiamo detto che questo lavoro rientra in un progetto più ampio. Tuttavia per analizzare i risultati per ora puntiamo a sviluppare un'applicazione stand-alone. Quello che ci prefiggiamo è creare una demo che ci permetta le seguenti operazioni:

- importare da file delle mesh nei più vari formati conosciuti
- applicare il nostro algoritmo
- visualizzare un'anteprima dei risultati
- compararli in qualche modo con altri metodi
- esportare mesh, pesi e skeleton in formati (Cal3D) utili al framework globale.

L'applicazione è stata sviluppata in C/C++ sfruttando fortemente l'STL (Standard Template Library). Oltre a questa ci si è avvalsi dell'ausilio di alcune librerie che esamineremo brevemente di seguito. La demo è stata compilata a 32 bit per testarla su diversi computer ed è stata sviluppata principalmente con Windows7 Professional a 64bit con Visual Studio 2010 Ultimate. Tuttavia, tutte le librerie utilizzate sono open-source e cross-platform.

4.1 Librerie

Per eseguire le operazioni sopra citate abbiamo sfruttato le seguenti librerie: Assimp, Ftlk, Pinocchio, Cal3D.

4.1.1 ASSIMP - importazione

Per l'importazione di mesh abbiamo usato le Assimp (Gessler, et al.). Queste sono librerie open sources C++ in grado di caricare una grande quantità di formati di scene tridimensionali. Non sono infatti pensate solo per caricare singole mesh umanoidi, ma anche scene più complesse. Tuttavia noi ci limiteremo ad utilizzarle per il caricamento di mesh rappresentanti personaggi.

Una scena viene caricata tramite il metodo `importer.ReadFile` di un oggetto `Assimp::Importer`. Tale metodo prende come input il nome del file con un bitflag che specifica il tipo di caricamento ed inizializza un puntatore ad un oggetto `aiScene` che conterrà la scena. L'oggetto `aiScene` contiene quindi un array di `aiMesh`, cioè una lista delle mesh della scena. Ogni `aiMesh` contiene una lista di vertici `aiVector3D`, e di facce `aiFace`. Oltre a queste informazioni `aiMesh` contiene per ogni vertice le normali, le coordinate di texture, i colori ecc. Come spiegheremo più avanti, queste altre informazioni possono essere tralasciate nello sviluppo del nostro algoritmo che sfrutta solo le coordinate dei vertici e le facce; verranno poi riprese solo al momento dell'esportazione della mesh.

4.1.2 PINOCCHIO - strutture e sviluppo algoritmo

Sfruttiamo la libreria Pinocchio per vari scopi. Anzitutto tale libreria permette di effettuare il rigging tra mesh e skeleton, cosicché siamo sicuri che lo skeleton sia posto all'interno della mesh. Pinocchio garantisce il successo del rigging se la mesh è posizionata in piedi davanti alla camera d'osservazione (non necessariamente in T-

pose). Nella nostra implementazione effettuiamo un piccolo controllo che verifica e corregge euristicamente tale condizione. Nel caso poi non venga fornito uno skeleton Pinocchio ne fornisce uno proprio.

Inoltre proprio grazie al rigging riusciamo ad effettuare un partizionamento secondo il metodo che vogliamo implementare, come descritto in precedenza.

Oltre a tutto ciò Pinocchio implementa al suo interno diversi metodi e strutture per gestire geometrie che ci fanno comodo nello sviluppo della nostra applicazione.

L'altro aspetto che ci torna utile è che Pinocchio fornisce un'implementazione del metodo basato su diffusione del calore che dà ottimi risultati. Per applicare tutti i suoi passi con successo Pinocchio vuole mesh manifold, completamente chiuse e ben orientate, ossia rivolte verso l'osservatore.

Come abbiamo detto Pinocchio contiene anche una demo chiamata DemoUI, sviluppata tramite Ftlk ed OpenGL, in grado di aprire una finestra per mostrare alcune animazioni (in un suo formato interno) applicate ad una mesh tramite LBS con quaternioni e pesi calcolati da Pinocchio stesso, con il metodo dell'heat diffusion. Manteniamo questa parte della demo nel nostro tool con gli stessi comandi di tastiera e mouse visti in precedenza ed ancora utilizzabili.

4.1.3 FTLK - interfaccia

Decidiamo quindi di sviluppare la demo di Pinocchio fornendo un'interfaccia grafica minimale che ci permetta di selezionare da dialog la mesh e l'animazione, che ci permetta di visualizzare un'anteprima dell'animazione con i pesi calcolati e che ci permetta di esportare mesh e skeleton nel formato che ci interessa. Giacché un visualizzatore d'anteprima è già implementato in Ftlk (FTLK) e liberamente disponibile nella demo di Pinocchio, decidiamo di utilizzare tali librerie per lo sviluppo dell'interfaccia grafica.

L’interfaccia prevederà tre campi di input di testo: un campo di selezione della mesh da caricare inizializzabilie tramite dialog, un campo per indicare il nome con cui salvare la mesh esportata, ed un campo in cui selezionare l’animazione scelta per testare i pesi anch’esso inizializzabile tramite dialog. Ci sarà anche una coppia di radio-button per decidere il tipo di formato in cui esportare la mesh (se binario o xml-based), una coppia per il tipo di formato dello skeleton (se binario o xml-based) ed infine una coppia per scegliere il metodo di calcolo dei pesi scelto (Atlas-based o tramite diffusione del calore implementata in Pinocchio). Oltre a questo vi sarà un bottone per visualizzare la finestra di preview utilizzata da Pinocchio ed un altro bottone per effettuare l’exporting nei formati Cal3D.

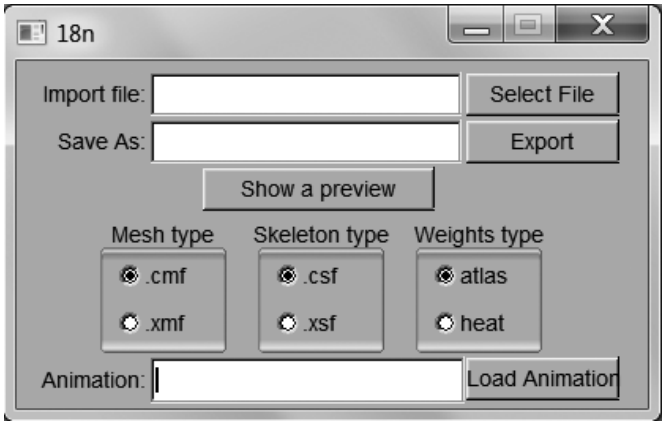


Figura 4.1: *interfaccia finale della demo sviluppata.*

4.1.4 CAL3D - esportazione

L’applicazione più generale con cui dovrà interagire il nostro progetto utilizza una libreria open source orientata all’animazione, nota nell’ambiente, chiamata Cal3D (Cal3D). Tale libreria ha un suo formato interno per la gestione di mesh e skeleton, che può essere sia di tipo binario sia xml-based. Il formato delle mesh contiene anche un campo per ogni vertice in cui è possibile inserire i pesi che abbiamo calcolato. Sfruttiamo quindi tale libreria per esportare mesh e skeleton nei formati che ci interessano.

4.1.5 Integrazione tra le librerie

Ovviamente anche Pinocchio nella sua demo ha dei metodi di importazione di mesh, ma solo per 5 formati di file: .obj, .ply, .off, .gts, .stl . Inoltre, nell'importare le mesh nelle sue strutture interne, non è in grado di caricare diverse informazioni come ad esempio le coordinate di texture. Per questo abbiamo preferito appoggiarci alle Assimp per il caricamento delle mesh. Tuttavia in uno o due casi testati le Assimp hanno fallito il caricamento di file, nonostante fossero in formati supportati (nello specifico .obj). Inoltre le Assimp non sono in grado di supportare file .gts mentre Pinocchio si. Abbiamo dunque deciso di tentare il caricamento principalmente tramite le Assimp, ma in caso queste falliscano, di tentare nuovamente il caricamento tramite l'importer di Pinocchio.

Assimp memorizza le mesh nelle strutture viste che sono adatte al rendering della mesh, ma sono meno adatte alla navigazione topologica della mesh. Per esempio, data una faccia, per ritrovare le facce adiacenti dovremmo cercare tra tutte le altre facce quelle che hanno un'edge in comune con la faccia corrente, il che può essere un procedimento complesso oltre che lento in una struttura semplicistica come quella delle Assimp. Esistono molte strutture più adatte alla trattazione topologica della mesh, che esplicitano o meno le diverse possibili relazioni di adiacenza tra gli elementi della mesh. Queste relazioni sono definite sui vertici (V), sugli edge (E) e sulle facce (F) e sono scritte con una notazione del tipo A-B (con A = {V,E,F} e B = {V,E,F}). Tale notazione esprime che dato un elemento di tipo B, si ottengono tutti gli elementi di tipo A adiacenti a B (per esempio la relazione E-F indica tutti gli edge di una faccia). Pinocchio implementa una sua versione personalizzata di queste strutture, quindi effettuiamo una conversione dalla mesh rappresentata in Assimp in una mesh rappresentata in Pinocchio.

Una volta caricato un file tramite Assimp in generale non vi è garanzia che tale file contenga una sola mesh, tantomeno che le singole mesh siano completamente connesse. Addirittura ogni faccia potrebbe riferire ad una propria copia diversa dello

stesso vertice e quindi la ricostruzione della topologia porterebbe a tante facce tutte topologicamente separate.

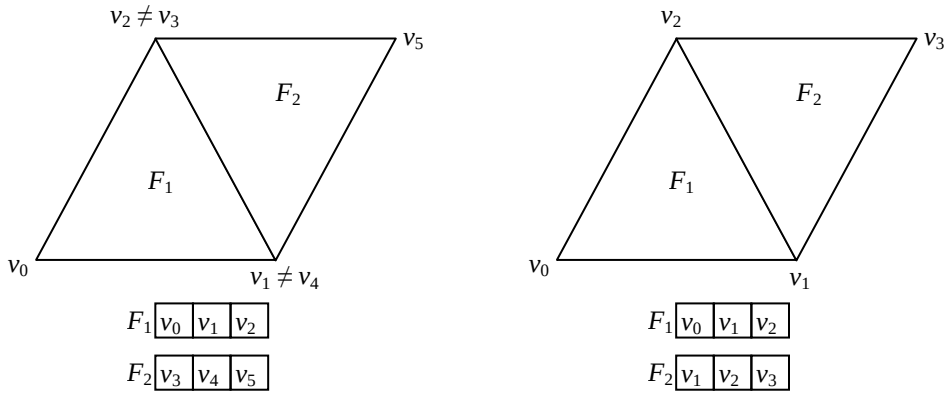


Figura 4.2: *a sinistra abbiamo una situazione in cui le due facce sembrano connesse, ma i loro vertici che formano l'edge che dovrebbe essere in comune tra le 2 facce sono in realtà diversi. A destra vi è la situazione che vorremmo in cui i vertici dell'edge su cui incidono le facce sono effettivamente gli stessi.*

Per ovviare a questo problema procediamo in due fasi. Fondiamo assieme vertici che possono essere tra loro incidenti (questo non è il modo migliore di procedere, ma è il più semplice e per ora ci basta; un approccio migliore fonde gli edge sfruttando una struttura di *suddivisione spaziale* come un *kd-tree* o *oct-tree*[18]). Per stabilire se due vertici sono tali guardiamo prima tutte le facce non degeneri, cioè che hanno i tre vertici ad una distanza tra loro maggiore di un certo epsilon (usiamo la funzione `numeric_limits<double>::epsilon()` della STL C++); prendiamo quindi tra tutte le facce non degeneri la lunghezza dell'edge più corto e la memorizziamo in una variabile `shortestEdgeLength`, dopo di che fondiamo assieme tutti i vertici che hanno una distanza tra loro inferiore a `shortestEdgeLength`.

[18] Le strutture di suddivisione spaziale dividono appunto lo spazio della scena in celle e sottocelle organizzandole in una gerarchia ad albero. Nelle foglie sono contenuti gli elementi immersi nello spazio suddiviso. Cosi per trovare un elemento più prossimo ad esempio ad un punto si può scendere nell'albero scegliendo la cella figlia che contiene il punto. Raggiunta poi una foglia si possono esaminare le sole celle adiacenti ad essa per trovare quale elemento contenuto nella struttura è più vicino al punto. Si riducono cosi i tempi di ricerca e di confronto. In un oct-tree lo spazio è suddiviso con piani che spezzano una cella sempre a metà sui 3 assi ottenendo cosi sempre 8 celle figlie. Un kd-tree invece usa piani si ortogonali agli assi, ma non necessariamente posizionati a metà della cella ed ogni volta si usa un singolo piano di taglio: una cella viene divisa in solo due celle figlie.

Pinocchio non ha strutture per sottomesh, ma tiene un unico oggetto `Mesh` composto di un vettore STL di `MeshVertex` che sono i vertici della mesh (`vector<MeshVertex>`) ed un vettore STL di `MeshEdge` che sono gli half-edges della mesh (`vector<MeshEdge>`). Osserviamo che in generale gli edges di una qualsiasi mesh potrebbero essere visti come un grafo non orientato. Sappiamo anche che un grafo non orientato può essere rappresentato tramite un grafo orientato, inserendo per ogni arco non orientato 2 archi orientati in direzioni opposte. Gli half-edges (semi-edge) sono proprio questi archi orientati. Inoltre ogni 3 half-edge consecutivi nel vettore costituiscono l'ordine di visita in senso orario degli edge di una faccia; in altre parole ogni 3 elementi in `vector<MeshEdge>` formano una faccia. Un `MeshVertex` contiene la posizione del vertice in un oggetto `Vector3` (il quale descrive un punto nello spazio 3D tramite 3 coordinate), la normale del vertice in un altro `Vector3` ed un intero che riferisce ad uno degli half-edge, incidenti sul vertice, che escono dal vertice stesso verso un altro vertice adiacente. Un `MeshEdge` contiene:

- l'indice `vertex` del vertice a cui punta,
- un indice di un altro half-edge che è il suo `twin` (il suo gemello); due half-edge che sono tra loro `twin` formano assieme un edge completo,
- un indice `prev` che riferisce al suo half-edge predecessore secondo l'ordine di visita orario degli edge della faccia.

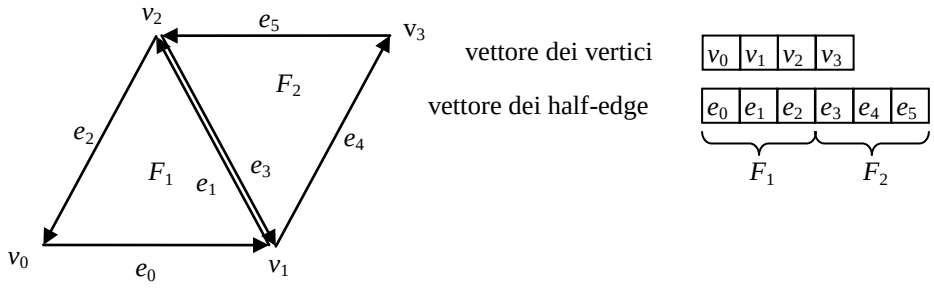


Figura 4.3: *gli half-edge e_i possono essere visti come delle frecce che definiscono i bordi; l'edge e_0 è il predecessore (il prev) di e_1; e_1 è il twin di e_3 e viceversa; v_2 è il vertice riferito da e_1. Nel vettore di half-edge ogni tre elementi abbiamo una faccia. Si noti come l'ordine di orientamento degli half-edge per le due facce F_1 ed F_2 sia sempre lo stesso per entrambe (come per tutte le altre facce della mesh).*

Strutture simili a questa, con qualche variante, sono tipicamente usate per rappresentare efficientemente la topologia di mesh manifold.

Nella seconda fase mettiamo dunque assieme tutte le facce Assimp convertendole in half-edge in un unico vettore di una mesh di Pinocchio. Come dicevamo nella mesh di Pinocchio non vi sono neppure strutture per memorizzare le altre informazioni della mesh, come le texture ad esempio. Manteniamo quindi la mesh Assimp originale assieme ad un mappaggio tra i vertici della mesh Assimp e i vertici della mesh di Pinocchio.

A questo punto nell'applicare il nostro algoritmo non facciamo alcun controllo che ci sia un'unica mesh totalmente connessa e chiusa. Potrebbero quindi esserci sottomesh sconnesse. Vedremo infatti che nelle prime fasi di individuazione delle regioni del nostro algoritmo, se sono presenti delle mesh (semi-)sconnesse, per esse si formeranno dei sottografi sconnessi. Questo risolverà di per sé il problema, perché ci rifaremo al caso di nodi-sfere appartenenti a grafi sconnessi, che saranno trattati in maniera rigida quando le mesh sconnesse non sono attraversate dallo skeleton.

Per il momento non ci siamo interessati a modificare anche Pinocchio in modo che accetti qualsiasi genere di mesh, né abbiamo implementato metodi che adattassero le mesh alle esigenze di Pinocchio, quindi nel caso si voglia utilizzare il metodo della diffusione del calore devono essere rispettate le stesse condizioni imposte dalla documentazione fornita dalla demo Pinocchio stesso (mesh manifold totalmente connesse e chiuse). Pinocchio contiene al suo interno un paio funzioni di test per queste proprietà. Inoltre prima di applicare le sue funzioni Pinocchio normalizza la dimensione del bounding box allineato con gli assi[19] della mesh.

[19] Il bounding box di una mesh è il più piccolo parallelepipedo che contiene l'intera mesh. Esso può avere allineato con gli assi (AABB: Axis Aligned Bounding Box) o meno.

Quando si esporta la mesh nei formati Cal3D si prendono le informazioni su vertici e facce dalla mesh di Pinocchio. A queste si aggiungono le informazione precedentemente tralasciate e conservate nelle mesh Assimp, sfruttando il loro mappaggio sulla mesh di Pinocchio salvato in precedenza.

Infine anche lo skeleton viene convertito in una rappresentazione a quaternioni ed esportato nel formato delle Cal3D.

4.2 Fase 1: implementazione della decomposizione

Come accennato nel capitolo precedente, Pinocchio oltre a fornirci delle sfere interne utili per determinare il partizionamento ci dà anche delle strutture su queste sfere che possiamo sfruttare per velocizzare i conti. In particolare queste sono:

- **Campo di Distanza**. Le sfere sono generate tramite un campo di distanza che, dato un punto, approssima la sua distanza dalla superficie. Grazie a tale campo siamo in grado di generare sfere in qualsiasi punto della mesh. Inoltre se il punto è interno alla mesh il campo di distanza restituisce un valore negativo, mentre se è esterno dà un valore positivo. Quindi, grazie a ciò, si è in grado di verificare se un generico punto è all'interno o all'esterno della mesh.

  Nella libreria il campo di distanza è costruito tramite la funzione `constructDistanceField` che prende come parametro un oggetto `Mesh` di Pinocchio e restituisce un puntatore ad un albero `TreeType`. Questo è la radice di un oct-tree. Passando un punto al suo metodo `locate` questo scende ricorsivamente nell'oct-tree sino ad una foglia. Richiamando poi il metodo `evaluate` di tale foglia e passandogli nuovamente il punto si ottiene la distanza approssimata dalla superficie. Poiché come abbiamo detto il bounding box della mesh è stato normalizzato, le distanze ritornate da questo metodo saranno tutti valori tra 0 e 1.

- **Superficie Mediale Approssimata**. Questa definisce una superficie interna alla mesh. In particolare Pinocchio genera la superficie mediale come un insieme di punti tra loro allineati e mediamente equidistanti dalla superficie.

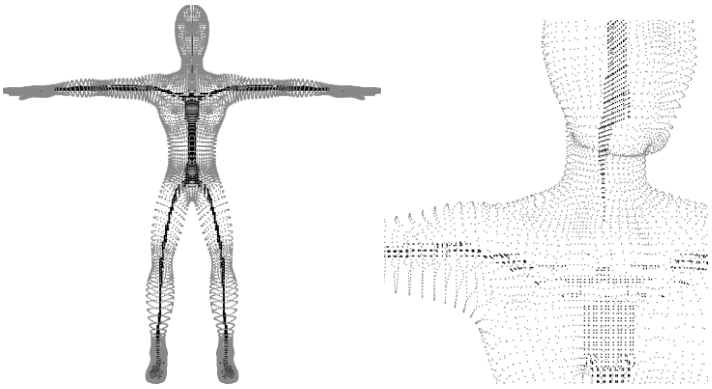


Figura 4.4: *in nero vi sono i punti della superficie mediale, in grigio i vertici della mesh.*

Oltre a definire una superficie interna ridotta Pinocchio determina anche un raggio (preso appunto come la minima distanza dalla superficie, calcolata grazie al campo di distanza) per ogni punto della superficie mediale. Dunque ogni punto della superficie mediale diventa una sfera, una bounding-sphere appunto. Queste sfere riempiono quasi completamente il volume della mesh, a differenza dello skeleton (a riguardo si ripensi anche al discorso attinente la perdita di volume nel passaggio da uno scheletro reale ad uno degli skeleton con i quali lavoriamo di solito a fine paragrafo 3.6.1).

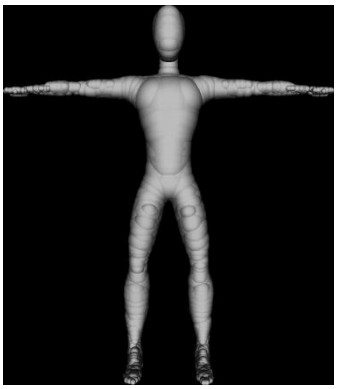


Figura 4.5: *sfere mediali: ogni sfera ha come centro un punto della superficie mediale e come raggio la distanza minima dalla superficie.*

Nella libreria il campo di distanza è un vettore della STL di oggetti `Sphere`, definiti in Pinocchio tramite il centro della sfera e il suo raggio. Il vettore di sfere mediali è ritornato dalla funzione `sampleMedialSurface` che prende come unico input il campo di distanza.

- **Grafo interno di nodi-sfera**. Da queste bounding-spheres Pinocchio estrae quelle di volume maggiore che inglobano le altre e le inserisce in un `vector<Sphere>`.

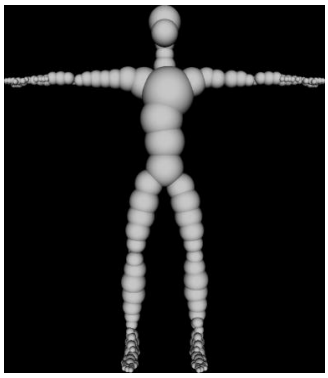


Figura 4.6: *sfere maggiori (nodi-sfera) estratte tra le sfere mediali.*

Si uniscono poi i centri di tali sfere tramiti degli archi, assicurandosi che tali archi rimangano all'interno della mesh. Si genera così un grafo i cui nodi sono le sfere (nodi-sfera) di dimensioni dominanti nella mesh e che individuano le aree principali della mesh stessa.

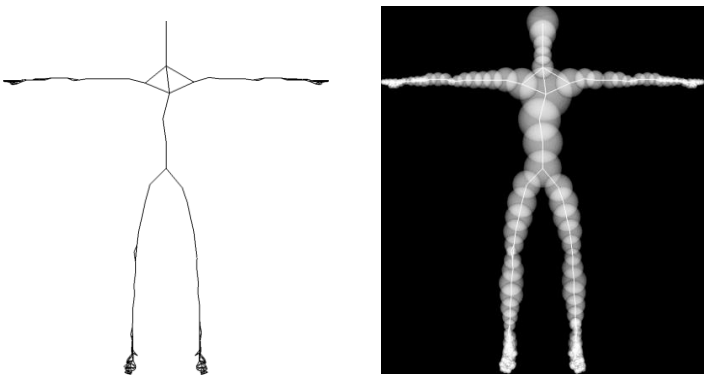


Figura 4.7: *grafo di nodi-sfera. A sinistra c'è solo il grafo, a destra il grafo con le sfere.*

Nella libreria il grafo viene generato passando alla funzione `connectSamples` il campo di distanza e il vettore delle sfere mediali; tale funzione ritornerà un oggetto `PtGraph`, il quale conterrà, oltre ad un metodo `bool integrityCheck()` di controllo dell'integrità del grafo, un vettore di vertici del grafo `vector<Vector3> verts` e gli edge descritti tramite lista di adiacenza `vector<vector<int>> edges`. La lista di adiacenza è un vettore che per ogni indice di vertice *i* contiene a sua volta una vettore di indici dei vertici adiacenti ad *i* stesso.

Sfruttando queste semplici strutture possiamo generare il nostro partizionamento in modo semplice e computazionalmente veloce.

4.2.1 Decomposizione - implementazione

Come prima cosa generiamo le strutture introdotte sopra con le funzioni che abbiamo descritto. Abbiamo quindi un grafo di sfere.

Per ogni nodo-sfera del grafo cerchiamo un bone principale (main-bone), se esiste. Teniamo quindi un vettore di indici che ha tanti elementi quanti sono i nodi-sfera del `PtGraph` generato e che per ogni nodo-sfera indica il main-bone trovato. Per riempire tale vettore procediamo con alcuni tentativi: per ogni nodo-sfera controlliamo se la sua sfera interseca qualche bones ed in tal caso, tra i vari bones intersecanti, prendiamo quello più vicino al centro del nodo-sfera come main-bone.

Può tuttavia accadere che la nodo-sfera non intersechi nessun bone; questa situazione capita per esempio in sporgenze molto evidenti della mesh. Ad esempio, se nello skeleton non ci sono i bones della mano, sulle punta delle dita ci saranno sfere che non toccano alcun bone.

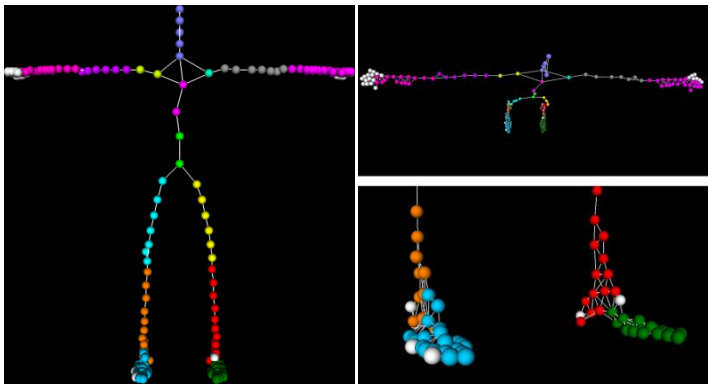


Figura 4.8: *le sfere bianche sulla punta delle mani e sui piedi non intersecano alcun bone.*

Per tentare di dare un main-bone anche a queste sfere procediamo come segue. Applichiamo Dijkstra partendo dalle sfere del grafo che hanno main-bone e sono connesse a sfere senza main-bone. Notiamo come la minima distanza tra un nodo-sfera senza main-bone e un nodo-sfera con main-bone è la stessa nelle due direzioni (poiché Dijkstra trova i cammini minimi e gli edge in realtà non sono orientati). Infatti, partire da nodo-sfere con main-bone e raggiungere quelle senza, equivale a partire da ogni nodo-sfera senza main-bone, raggiungere tutte quelle con main-bone e prendere la minima tra queste distanze. In questo modo applichiamo Dijkstra una sola volta, anche se da sorgente multipla. Questo ragionamento ci tornerà utile anche quando estenderemo le regioni dai loro bordi. A questo punto per ogni nodo-sfera senza main-bone *s*, sappiamo quale è il nodo-sfera con main-bone *t* più vicino ad *s*. Assegniamo come main-bone di *s* lo stesso main-bone di *t*.

Come accennato, anche se in modo estremamente raro, può capitare che Pinocchio generi un grafo non completamente connesso. Questo capita quando ci sono parti di mesh quasi o completamente chiuse a formare sacche a sé stanti. Ad esempio capita se il personaggio ha un marsupio o un cappuccio semichiuso. Le nodo-sfere interne ad una sacca formeranno un grafo a sé stante, sconnesso dal resto del grafo e a nessuna di esse riusciremo ad assegnare un main-bone, poiché nessuna di esse intersecherà alcun bone né sarà raggiungibile tramite Dijkstra. In tal caso indichiamo tutte le nodo-sfere del sottografo come rigide e forziamo il loro main-bone

prendendo quello della sfera più prossima nel grafo principale (dove il grafo principale è quello con main-bone assegnate ad ogni sua nodo-sfera tramite i passi precedenti). Osserviamo che in questo passo sfruttiamo la distanza Euclidea per superare il problema della disconnessione. Ricordiamo infatti che un vantaggio dei metodi basati sulla distanza Euclidea è che le connessioni sono definibili a prescindere. Tuttavia noi sfruttiamo tale proprietà solo per parti sconnesse e soltanto per trattarle in modo rigido.

Alla fine di tutti questi passi otteniamo un main-bone per ogni singola nodo-sfera.

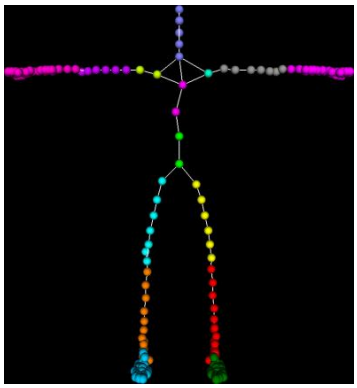


Figura 4.9: *i nodi-sfera sono colorati con un colore diverso a seconda del loro main-bone.*

A questo punto cerchiamo i main-bone per ogni vertice. Per prima cosa troviamo per ogni vertice la sfera della superficie mediale più vicina ad esso: per ogni sfera prendiamo la distanza tra il vertice e il centro della sfera sottraendo il raggio della sfera mediale stessa; scegliamo quindi la sfera che ha tale distanza minima tra tutte. Cosi ad ogni vertice corrisponde una singola sfera della superficie mediale.

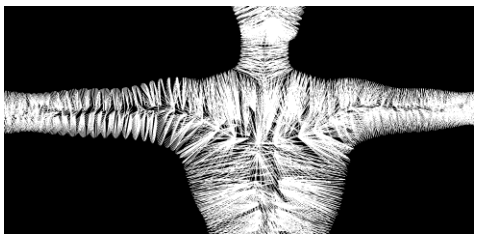


Figura 4.10: *colleghiamo ogni vertice ad un punto della superficie mediale.*

In questo modo è come se ci muovessimo da ogni vertice ad un punto della superficie mediale.

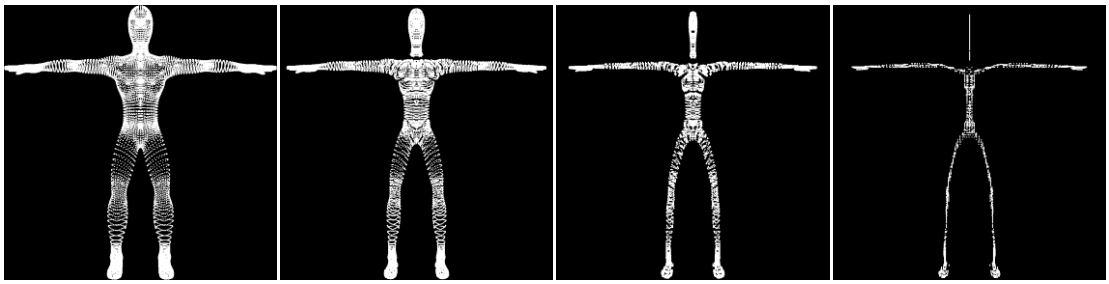


Figura 4.11: *da ogni vertice ci muoviamo verso un punto interno della superficie mediale.*

Valutiamo poi se la sfera-mediale trovata interseca qualche bone e in questo caso prendiamo il bone più vicino come main-bone del vertice. Se ciò non avviene, dalla sfera-mediale passiamo ad una nodo-sfera del grafo, prendendo la nodo-sfera più vicina alla sfera-mediale. Assegniamo come main-bone del vertice lo stesso main-bone della nodo-sfera. Se la nodo-sfera è stata settata come rigida al passaggio precedente, segniamo che il vertice si comporta in modo rigido su quel bone: ovvero il peso per il main-bone del vertice verrà forzato ad 1, mentre per tutti gli altri bones sarà resettato a 0. Abbiamo così stabilito un main-bone, o, se vogliamo, un bone che ha influenza maggiore per ogni vertice.

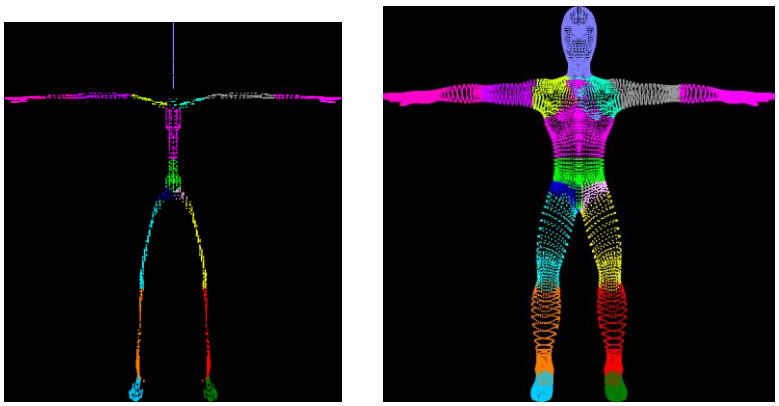


Figura 4.12: *nelle due figure ogni punto è colorato con un colore diverso a seconda del suo main-bone. A sinistra da ogni vertice della superficie mediale possiamo ricavare un main-bone. Questo si ripercuote a destra sui vertici più prossimi ai punti della superficie mediale. In questo modo si definisce un partizionamento della mesh.*

Poiché nel nostro metodo, diversamente da quello originale, ogni regione corrisponde esattamente ad un bone (e viceversa), i main-bone trovati ci permettono di partizionare i vertici in regioni distinte: di fatto un main-bone identifica una regione. L'aver utilizzato le sfere interne ed esserci mossi tramite Dijkstra ci garantisce di aver trovato come main-bone il bone più prossimo al vertice, rispettando comunque la geometria. Il costo complessivo dipende principalmente dalla velocità con cui si creano le sfere della superficie mediale e quindi è pari alla velocità del rigging di Pinocchio stesso.

4.2.2 Rilevamento del confine - implementazione

Dobbiamo ora determinare i boundaries, ovvero le sequenze concatenate di punti di confine che formano un bordo. Come abbiamo detto gli edge di confine sono semplici da identificare. Scorriamo tutti gli edge e controlliamo se i loro vertici sono in 2 regioni diverse R_i ed R_j. Poiché ad ogni regione corrisponde esattamente un bone (rispettivamente R_i ha bone b_i ed R_j ha bone b_j) basta controllare se i due vertici hanno main-bone diverso. Quando troviamo un edge di confine calcoliamo il punto di confine. Abbiamo detto che nel cercare tale punto tentiamo di bilanciare le distanze del punto da b_i e da b_j, rimanendo sempre sull'edge. Per fare ciò sfruttiamo un semplice gioco di proporzioni. Prendiamo i 2 vertici estremi v_0 e v_1 dell'edge: per ognuno guardiamo la distanza da b_i e da b_j, ottenendo $d(v_0,b_i)$, $d(v_0,b_j)$, $d(v_1,b_i)$ e $d(v_1,b_j)$. Scegliamo una delle due bone, per esempio b_i e partiamo da uno dei due vertici, per esempio v_0. Supponiamo che questo sia il vertice più vicino a b_i, cioè $d(v_0,b_i) < d(v_1,b_i)$. Dalle distanze possiamo calcolare una percentuale che ci dice quanto v_0 e v_1 sono prossime a b_i; tale valore per v_0 è pari a:

$$\text{involved}(v_0,b_i) = d(v_0,b_j) \,/\, (d(v_0,b_i)+d(v_0,b_j))$$

Si noti che al numeratore si valuta la distanza da b_j, poiché più v_0 è vicino a b_i più la distanza da b_j è maggiore.

Stessa cosa per v_1:

$$\text{involved}(v_1,b_i) = d(v_1,b_j) / (d(v_1,b_i)+d(v_1,b_j))$$

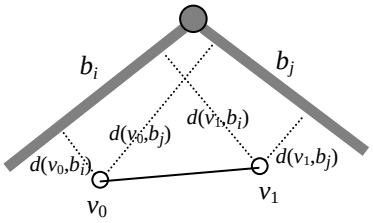


Figura 4.13: *in figura è mostrato il significato geometrico di involved.*

Ora se entrambi i valori sono maggiori di 0,5 o minori di 0,5 , il punto perfettamente bilanciato che vorremmo (cioè perfettamente equidistante dai 2 bones) giace lungo la retta che passa per l'edge, ma al di fuori dell'edge stesso. Come punto di confine scegliamo quindi tra v_0 e v_1 quello più bilanciato: se $d(v_0,b_i)$ - $d(v_0,b_j)$ è più vicino a 0 di $d(v_1,b_i)$ - $d(v_1,b_j)$ scegliamo v_0, altrimenti scegliamo v_1.

Se invece i valori sono uno maggiore di 0,5 e l'altro minore di 0,5 possiamo stabilire un punto intermedio sull'edge. Scegliamo di partire ad esempio da v_0. Prendiamo il vettore direzione *dir*, che va da v_0 a v_1. Memorizziamo la lunghezza di tale vettore nella variabile `length`. La differenza tra involved(v_0,b_i) e 0.5 che salviamo nella variabile `mov` ci dice quanto ci manca da v_0 per arrivare ad una percentuale di distanze bilanciata. La differenza tra involved(v_0,b_i) e involved(v_0,b_j) che salviamo nella variabile `maxMov` ci dice di quanto la percentuale cambia muovendoci lungo tutto il vettore direzione *dir*, ovvero per tutta la lunghezza `length`. Quindi per bilanciare le distanze dobbiamo muoverci da v_0 di una distanza (che salviamo nella variabile `newLength`) calcolabile con un semplice gioco di proporzioni[20]:

[20] Può capitare che v_0 e v_1 siano equamente distanti dalle 2 bones. In questo caso l'intero edge è collineare al bordo che cerchiamo. Poiché per poter poi riconnettere tra loro tutti i punti bordo dobbiamo comunque mantenere l'edge di confine che ha generato il punto di confine, in questo caso prendiamo per semplicità il punto esattamente medio tra v_0 e v_1.

```
maxMov/length = mov/newLength        ⇒      newLength = length*mov/maxMov
```

Dobbiamo ancora ricostruire i bordi connettendo sequenzialmente i punti di confine. Abbiamo detto che generiamo un punto di confine da ogni edge di confine e che da tale edge sappiamo risalire alle due regioni che l'edge attraversa. Quindi per ognuna di queste 2 regioni salviamo tale punto di confine assieme all'edge di confine che l'ha generato. Per ricostruire i bordi della regione scorriamo quindi gli edge di confine. Se l'edge non è stato visitato lo segniamo come tale e ci muoviamo iterativamente da questo verso un suo edge adiacente che sia anche esso di confine. Creiamo così una concatenazione di edge di confine. La stessa concatenazione applicata ai punti di confine genererà uno dei bordi. Proseguiamo quindi se ci sono altri edge di confine non ancora visitati alla ricerca di altri possibili bordi.

4.3 Fase 2: implementazione dell'estensione

Passiamo ora a vedere l'implementazione del calcolo della distanza geodetica, cosa comporta il criterio di distanza di estensione che abbiamo adottato e come influisce sulla generazione dell'area sovrapposta.

4.3.1 Calcolo della distanza geodetica - implementazione

Per quanto riguarda il calcolo della distanza geodetica non abbiamo implementato uno degli algoritmi suggeriti nello studio originale del metodo (Lu, et al., 2008). Per il momento ci siamo limitati ad applicare Dijkstra classico (approssimazione debole della distanza geodetica, come spieghiamo in APPENDICE C) trasformando la mesh in liste di adiacenza e rimandando semmai lo sviluppo in uno dei metodi più raffinati. Questa scelta è stata effettuata perché anche se Dijkstra classico dà risultati meno precisi, la rappresentazione tramite liste di adiacenza ci permette di superare l'ostacolo di vertici non manifold e bordi aperti. Ricordiamo infatti che nel progetto

più globale di cui fa parte questo lavoro di ricerca, le mesh vengono acquisite on-line tramite un dispositivo di scansione tridimensionale. È dunque facile che nell'acquisizione e ricomposizione la mesh presenti vertici manifold o bordi aperti. La rappresentazione della mesh in Pinocchio (e in molte delle librerie che trattano efficientemente la topologia della mesh) permette di navigare da una faccia alla sua adiacente tramite l'attributo `twin` degli half-edge. Tuttavia la presenza di vertici non manifold può comportare che più di due half-edge siano tra loro incidenti e quindi attraversando due volte un `twin` si potrebbe non ritornare all'edge di partenza come accade invece nelle mesh manifold.

In un'implementazione futura si può tentare di ripulire in qualche modo la mesh da situazioni non manifold e bordi aperti (problema trattabile da un punto di vista implementativo, ma non banale dal punto di vista del mantenimento dell'anatomia: potrebbe generare una sconnessione anatomicamente non coerente). Oppure si potrebbe adattare Dijkstra alla trattazione di facce sulle liste di adiacenza (con tempi di esecuzione leggermente maggiori rispetto a mesh manifold tenute in strutture classiche come quella di Pinocchio).

C'è un'altra differenza d'implementazione che adottiamo rispetto all'algoritmo originale. Come abbiamo spiegato, dato un vertice, nell'algoritmo originale gli autori scelgono il punto di confine più vicino c_k tramite distanza Euclidea, poi calcolano le distanze geodetiche separatamente per c_k e altri 4 punti di confine vicini ad esso ed infine scelgono la minima tra queste distanze. Dunque il costo di tale metodo è pari al numero di vertici moltiplicato 5 volte l'applicazione di un algoritmo di distanza geodetica. Noi, invece di partire dai vertici, partiamo dai punti di bordo. Infatti, muovendosi con Dijkstra classico in un grafo non orientato o con uno degli algoritmi più raffinati che si muovono sulla superficie, il percorso più breve tra un vertice ed un punto di confine coincide con il suo percorso inverso dal punto di confine al vertice. Inoltre invece di calcolare la distanza separatamente per tutti i punti bordo usiamo una versione dell'algoritmo che parte da sorgente multipla, il che è fattibile sia con

Dijkstra classico, sia con uno dei metodi di calcolo della distanza geodetica più raffinati. In questo modo consideriamo meno volte uno stesso cammino e il costo del calcolo delle distanze non va moltiplicato per il numero di punti bordo, ma solo per il numero di bordi.

In ogni caso per ora noi partiamo semplicemente dai punti di confine di ogni bordo e raggiungendo i vertici delle varie regioni tramite Dijkstra classico. Rispetto a Pinocchio quindi richiediamo che la mesh sia connessa non completamente, ma solo nelle parti attraversate dallo skeleton. Una volta completato il calcolo, oltre a stabilire la minima distanza per ogni vertice, risalendo l'albero dei padri, possiamo ricavare anche il bordo più vicino, cosa che ci tornerà utile per stabilire i pesi.

4.3.2 Calcolo della distanza di estendibilità - implementazione

Prendiamo ora ogni bordo. Scorriamo i suoi punti e calcoliamone la distanza dal bone della regione. Tale calcolo è estremamente semplice. Possiamo fare una media di queste distanze o prendere la minima, oppure fare una media pesata con lunghezza segmenti di bordo anziché semplicemente sui vertici. In questo modo troviamo il raggio di estendibilità partendo dal bordo e dirigendoci nei due lati opposti delle regioni. Per ora noi abbiamo deciso di prendere la distanza minima dei punti.

4.3.3 Generazione dell'area sovrapposta - implementazione

Anche in questo caso possiamo procedere esattamente come descritto per la versione generalizzata dell'algoritmo originale. Notiamo infatti che in quest'ultimo, anche se un bordo separa solo due regioni, muovendoci da esso potevamo comunque coinvolgere vertici appartenenti a più di due sole regioni. Si pensi ad esempio alla regione dell'inguine.

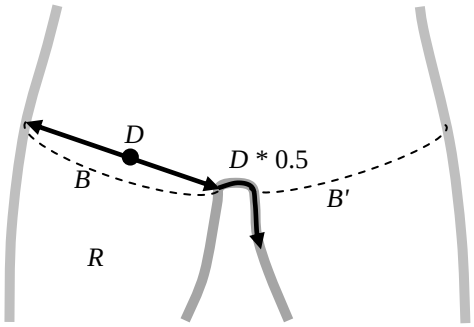


Figura 4.14: *nel metodo generalizzato originale, considerando la regione R relativa ad una coscia, partendo dal suo confine inguinale B e muovendoci fino alla massima distanza di estendibilità (D * 0.5) tramite distanza geodetica, in alcune direzioni oltrepassiamo anche l'altro bordo B', nonostante questo non sia un confine di R.*

Questo non è problematico per il calcolo finale dei pesi, anzi è una situazione frequente e trattata dall'algoritmo generalizzato stesso. Basti osservare come più di due regioni estese si sovrappongano molto spesso in mesh generiche.

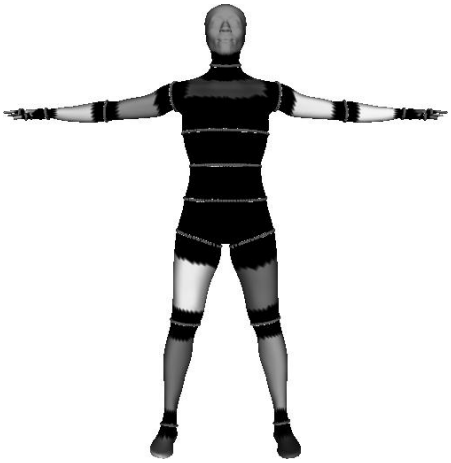


Figura 4.15: *le estensioni delle regioni sono le aree in nero; com'è visibile queste non sono ben separate tra loro, quindi un vertice può essere coinvolto da diverse regioni. Ad esempio sull'addome un vertice può essere influenzato da 3 regioni: quella dell'addome stesso, quella del torace inferiore e quella del bacino.*

4.4 Fase 3: implementazione del calcolo pesi

Abbiamo detto che sfruttiamo un metodo definito che per ogni vertice v usa la distanza $d_j(v)$ dal più vicino bordo $B'_{j,k}$ e la distanza tra tale bordo e il confine

interno che delimita la parte rigida. Quest'ultima distanza è il diametro $D_{j,k}$ associato a $B_{j,k}$.

Al passo precedente, con Dijkstra, abbiamo calcolato la distanza geodetica $\bar{d}_j(v)$ (ma potevamo usare un qualsiasi tipo di distanza), tra il vertice v e il bordo più vicino della regione R_j. Sempre tramite Dijkstra possiamo risalire a tale bordo $B_{j,k}$. Ora, invece di calcolare esplicitamente il bordo $B'_{j,k}$, ricordiamo che la distanza tra $B'_{j,k}$ e $B_{j,k}$ è pressoché pari alla distanza di estensione, cioè al raggio $r_{j,k} = D_{j,k} * 0.5$ e distinguiamo due casi: il primo in cui v appartiene ad R'_j ma non ad R_j ed il secondo in cui v appartiene ad R_j (e quindi anche ad R'_j).

- CASO I ($v \in R'$, $v \notin R$). Poiché la distanza da $B_{j,k}$ a v è pari a $\bar{d}_j(v)$, la distanza da v a $B'_{j,k}$ sarà pari a:

$$d_j(v) = r_{j,k} - \bar{d}_j(v)$$

- CASO II ($v \in R$). In questo caso la distanza tra v e $B'_{j,k}$ è pari alla distanza da v e $B_{j,k}$, cioè $\bar{d}_j(v)$, più la distanza da $B_{j,k}$ a $B'_{j,k}$:

$$d_j(v) = \bar{d}_j(v) + r_{j,k}$$

Ricordiamo infatti che la distanza $d_j(v)$ viene calcolata solo se v è dentro il diametro. Poiché per calcolare le sigma $d_j(v)$ va diviso per il diametro, possiamo semplificare i conti, in maniera simile a quanto abbiamo fatto per il primo metodo ed otteniamo:

$$\sigma_j(v) = \begin{cases} 1 & \text{se } D_{i,k} < \bar{d}_j(v) \\ 0{,}5 + \dfrac{\bar{d}_j(v)}{D_{i,k}} & \text{se } r_{i,k} \leq \bar{d}_j(v) \leq D_{i,k} \\ 0{,}5 - \dfrac{\bar{d}_j(v)}{D_{j,k}} & \text{se } 0 \leq \bar{d}_j(v) \leq r_{i,k} \end{cases}$$

Quindi in un certo senso ci muoviamo di nuovo dal bordo $B_{j,k}$ come nel primo metodo. Stavolta tuttavia lo facciamo solo per una regione rispetto al suo bordo più vicino e normalizziamo i pesi solo in seguito. Infatti $\sigma_j(v)$ riguarda solo la regione *j*, mentre ricordiamo che nel primo metodo la funzione $d_{i,j}(v)$ era calcolata per due regioni *i* e *j*.

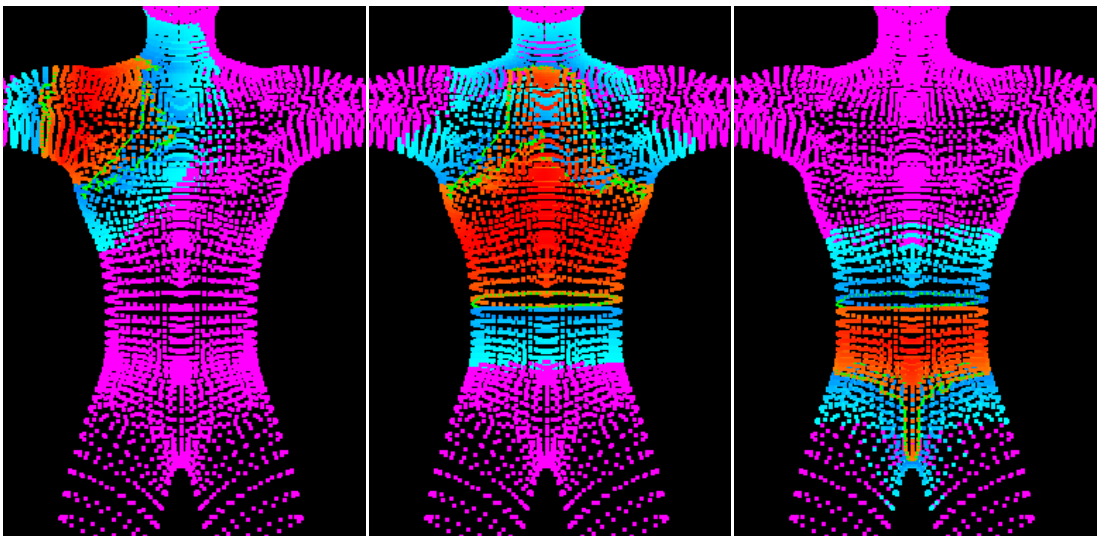


Figura 4.16: *la colorazione mostra qual è l'intensità dei σ_j per i vertici una regione j (e quindi per il bone j): i vertici fucsia non sono influenzati dalla regione considerata; la colorazione che va dal celeste all'azzurro indica vertici che hanno σ_j che varia da* 0 *a* 0,5*, mentre la colorazione che va dall'arancione al rosso indica vertici che hanno σ_j che varia da* 0,5 *a* 1*; in verde sono indicati i bordi individuati da cui parte l'estensione della regione.*

Dopo di ciò i pesi vengono normalizzati, come abbiamo detto in precedenza, tramite la formula:

$$w_j(v) = \frac{\sigma_j(v)}{\sum_{i \in I(v)} \sigma_i(v)}$$

La scelta di usare il diametro non è la soluzione più precisa possibile. Infatti la distanza di estensione da $B'_{j,k}$ a $B_{j,k}$ non è pari al raggio, o meglio è pari al raggio solo per angoli convessi, mentre è maggiore per angoli concavi.

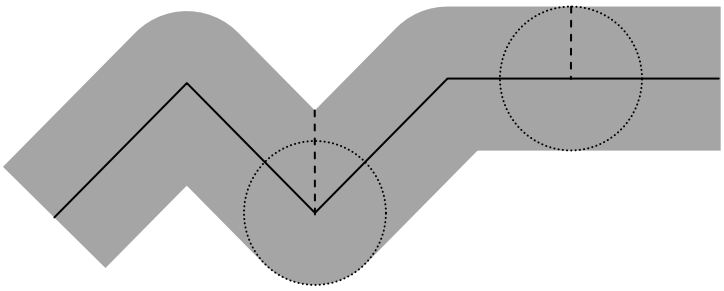


Figura 4.17: *la distanza dal bordo per angoli concavi non è pari al raggio.*

E questo vale anche per la distanza di estensione da $B_{j,k}$ verso l'interno della regione. Inoltre, se la regione è completamente coperta da tutte le aree di sovrapposizione, verso l'interno il punto di massima rigidità verrà schiacciato da queste e quindi in realtà la sua distanza sarebbe minore rispetto alla distanza di estensione.

Dunque usando $D_{i,k}$ al posto di δ_j usiamo ancora un'approssimazione non precisa. Tuttavia $D_{i,k}$, diversamente da δ_j, non dipende dalla grandezza della regione quindi non risente dei problemi descritti nel paragrafo 3.9 . Inoltre evitando di esplicitare i bordi interni ed esterni velocizziamo i calcoli.

Capitolo 5

Risultati

L'algoritmo è stato testato su diverse mesh umanoidi: principalmente le stesse testate da Pinocchio e scaricate dal sito ufficiale, più un paio di altre mesh reperite altrove. Tali mesh hanno circa 13k facce; la mesh di dimensioni maggiori raggiunge le 30k facce, mentre quella di dimensioni minori si aggira intorno a sole 900 facce. Si è utilizzato il viewer della demo di Pinocchio, integrato nella nostra demo, per avere una preview dei risultati.

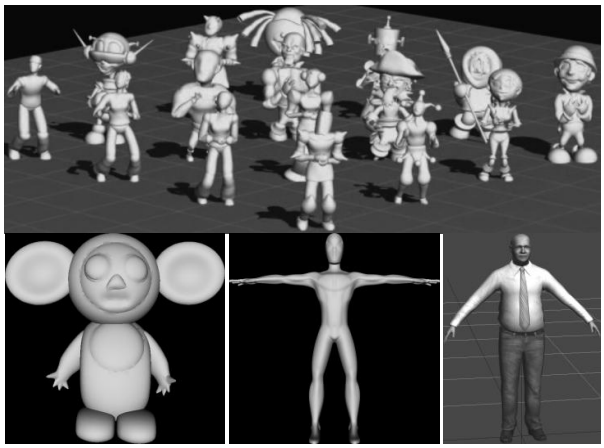


Figura 5.1: *sopra vi sono le mesh fornite da Pinocchio. Sotto: a sinistra un'altra mesh fornita da Pinocchio di forma e posizione molto particolare, adatta ad individuare problemi; al centro la mesh con maggior numero di facce che abbiamo testato, anche molto adatta a prestarsi come caso migliore dal punto di vista dei risultati; a destra una mesh con un vertice non-manifold su una caviglia, dei bordi aperti sugli occhi (che sono rappresentati da semi-sfere) e delle texture.*

Per analizzare visivamente i passi intermedi dell'algoritmo è stato introdotto del codice di debug, abilitabile a tempo di compilazione, che produce diversi files X3D, uno standard di descrizione di ambienti virtuali, non proprietario, basato su XML e sviluppato da Web 3D Consortium. Tali files sono visualizzabili tramite diversi tools (come ad esempio view3dscene) e sono facilmente manipolabili con editor testuali qualora si rendesse necessario, ad esempio per cambiare la colorazione. Tramite questi abbiamo ricavato molte delle immagini utilizzate per il debug dell'algoritmo e per molte delle figure usate in questo stesso documento. Per le stesse identiche ragioni sono stati prodotti anche alcuni files SVG (Scalable Vector Graphics), un altro standard basato ancora su XML e sviluppato da W3C che abbiamo usato per generare alcune immagini vettoriali bidimensionali di sezioni di figure o simili.

5.1 Ricerca di una metrica

Abbiamo anche cercato una qualche metrica matematica per stabilire la qualità dei risultati. Vi è uno studio di valutazione dei metodi di skinning (Jacka, et al., 2007) in cui si dispone di un certo numero di pose di altissima qualità che descrivono una stessa immagine in pose diverse. Scegliendo tra queste una posa di riposo si rigenerano nuovamente tutte le altre pose di cui già si dispone, ma stavolta tramite l'LBS. Per ogni posa avremo quindi una *posa di riferimento* ad alta qualità e una *posa approssimata* tramite l'LBS. Le approssimazioni sono quindi confrontate con quelle di riferimento usando due delle *Figures of Merit* (Silva, et al., 2005): la *deviazione geometrica*, che misura la distanza geometrica da ogni vertice della mesh approssimata al punto più vicino della mesh di riferimento, e la *deviazione normale*, che misura la differenza tra le normali dei corrispettivi vertici nelle pose approssimate e di riferimento. Dunque la deviazione geometrica ci fornisce una misura di quanto la forma di approssimazione risulta vicina all'originale, mentre la deviazione normale ci fornisce una misura importante a livello visivo poiché le normali sono usate nei

calcoli di illuminazione. Se la mesh approssimata ha una grande deviazione normale potrebbero generarsi artefatti visivi durante il rendering.

Purtroppo non abbiamo trovato qualcosa di già implementato e lo sviluppo di un tester simile richiederebbe un lavoro a parte. Potremmo applicare questo metodo solo alla posa di riposo, ma non avrebbe molto senso. Di seguito quindi ci limiteremo solo a valutazioni di carattere generale, ma individuando caratteristiche che possono comunque tornarci utili per sviluppi futuri.

5.2 Costi computazionali

Analizziamo intanto i costi del nostro nuovo algoritmo, prima stabilendo quale è l'ordine di grandezza del tempo richiesto e successivamente osservando alcuni risultati sperimentali sui tempi di esecuzione.

5.2.1 Analisi dei costi

Per analizzare i costi ricordiamo due cose. Sappiamo che il confronto nello spazio di un qualsiasi insieme di *m* elementi con un altro insieme di *n* elementi ha in generale costo *m* * *n*. Se si vuole ottimizzare il confronto si possono sfruttare strutture di ricerca spaziale che operano in tempo logaritmico, come ad esempio un kd-tree o un oct-tree, possibilmente costruite sul set più grande di elementi. In tal modo per ogni elemento del set più piccolo, per esempio quello di *n* elementi, si applicherebbe una ricerca logaritmica sul numero di elementi (cioè *m*) del set più grande, e dunque il costo si ridurrebbe a $O(n \log m)$. Di seguito teniamo quindi sempre da conto questa possibile ottimizzazione quando confrontiamo due insiemi di elementi. L'altro aspetto che ricordiamo è che i bones sono generalmente un numero estremamente ridotto, specialmente rispetto alla complessità della mesh. Questo perché lo skeleton serve proprio a semplificare il più possibile la trattazione di un numero elevato di vertici.

Dunque confrontare un set di elementi con i bones ha costo dipendente dal numero di elementi del set. Se poi usiamo anche in questi confronti strutture come kd-tree o simili il costo diventa davvero irrisorio. Stesso ragionamento vale per i joints.

Vediamo i costi computazionali per la prima fase di decomposizione. La prima ricerca di un main-bone per ogni nodo-sfera consiste nel confrontare le nodo-sfere con ogni bone. Per quanto detto sopra il costo di questa operazione dipende principalmente dal numero di nodi-sfera che è limitato. La quantità di sfere generabili non dipende tanto dalla quantità di vertici, quanto dalla complessità intrinseca della geometria. È stato osservato sperimentalmente che su superfici umanoidi tipicamente trattate vengono generate intorno a 200 sfere. Per mesh di forma più complessa il numero di queste sfere si aggira mediamente sotto le 500 sfere. In ogni caso lo stesso metodo `packSpheres` che genera i nodi-sfera, impone di non creare più di un numero costante di sfere (di default tale limite è impostato a non più di 1000 sfere). Dunque la prima ricerca ha costo assai trascurabile.

La seconda ricerca applica Dijkstra ai nodi-sfera prendendo solo quelli che non hanno ancora un main-bone e quelli di confine. Anche qui il costo è trascurabile, sia per il costo ottimale di Dijkstra, ma soprattutto ancora una volta per il numero (ulteriormente ridotto) di nodi-sfera valutati, oltre al fatto che per le figure che generalmente trattiamo, i nodi-sfera hanno solitamente una cardinalità molto bassa di archi incidenti nel grafo.

La terza ricerca trova i main-bones per nodi-sfera che appartengono a grafi secondari sconnessi (che andranno poi trattati rigidamente). Qui va fatto un confronto tra i nodi-sfera con main-bone e nodi-sfera senza main-bone. Dunque il costo è ancora minimo.

Si passa quindi a cercare un main-bone per ogni vertice. Come prima ricerca si passa dal vertice alla sfera mediale più prossima. Sperimentalmente il numero di sfere mediali in questo caso è all’incirca simile o inferiore al numero di vertici, ma può

anche essere maggiore per forme estremamente complesse. Tuttavia abbiamo osservato che tale numero resta comunque lineare nel numero dei vertici e, per immagini umane di forma tipicamente trattata, si riduce molto al disotto del numero di vertici.

Poiché per ogni vertice troviamo esattamente una sola sfera mediale, le fasi successive di confronto tra sfere mediali e bones e (se necessario) tra sfere mediali e nodi-sfera hanno ancora costo dipendente dal numero di vertici (sempre ottimizzabile tramite kd-tree).

L'individuazione dei punti di bordo e la ricostruzione dei bordi stessi è lineare nel numero di edge della mesh.

Il costo della decomposizione, nel suo complesso, risulta quindi dipendere dal costo della fase di rigging di Pinocchio com'è naturale che sia, dato che abbiamo scelto proprio di usare quel rigging per generare il partizionamento.

Passiamo ora alla seconda fase di estensione delle regioni. Il costo del calcolo del raggio di estensione per ogni bordo è pari al numero di vertici del bordo stesso. Il numero di bordi dipende dal numero di regioni generate, che nel nostro algoritmo equivale al numero di bones. Quindi il costo per trovare tutti i raggi di estensione è molto basso.

La fase di calcolo delle distanze geodetiche invece risulta una parte più costosa. Anche questa fase si applica per ogni bordo di ogni regione. Più precisamente, per ogni bordo applichiamo l'algoritmo verso l'interno e verso l'esterno della regione. In realtà si considera un'unica sorgente connessa poi a tutti i bordi, quindi si applica l'algoritmo di distanza geodetica due volte (partendo dal bordo verso l'interno e verso l'esterno) per l'intera regione. Il tempo dunque è pari al costo di Dijkstra applicato a tutti i bordi, che sono comunque pari al numero di joints non foglia e quindi un numero molto basso. Per ridurre ulteriormente il calcolo possiamo limitare la ricerca

di Dijkstra in profondità, portandoci dietro, su ogni vertice della frontiera, di quanto ancora possiamo avanzare prima di aver percorso tutta la distanza di estensione.

Stabilite le distanza geodetiche rispetto ai bordi della regione, nell'ultima fase calcoliamo i pesi in tempo lineare.

Riassumendo i punti chiave da cui dipende il costo complessivo dell'algoritmo sono: la decomposizione, il cui costo dipende dal metodo di rigging scelto (nel nostro caso quindi dal rigging di Pinocchio) e il calcolo delle distanze geodetiche che è pari al costo dell'algoritmo di Dijkstra.

5.2.2 Tempi d'esecuzione sperimentali

I tempi sono stati misurati tramite le funzioni di libreria `time.h` della STL calcolando la differenza tra il tempo di inizio di una fase ed il tempo al completamento della fase stessa. Nella pratica il collo di bottiglia dell'intero algoritmo si è dimostrato essere la creazione del campo di distanza generato durante il rigging di Pinocchio con la funzione `constructDistanceField` e di conseguenza anche il numero di sfere mediali generate usate nella fase di partizionamento dei vertici (ovvero la ricerca della sfera mediale più superficiale).

Nella descrizione del metodo originale nella sua prima versione (Lu, et al., 2008) l'algoritmo è stato eseguito su un Pentium IV da 1.66GHz con 1 GB di RAM ed applicato ad un modello umano con 14k facce, da cui si generavano 24 curve di confine e 25 regioni. Tale esecuzione impiegava circa 4 minuti per creare le zone di sovrapposizione e calcolare i pesi. Tramite diversi computer abbiamo verificato sperimentalmente i tempi del nostro algoritmo sulla mesh più grande di cui disponevamo usando lo skeleton generato da Pinocchio stesso. La mesh ha circa 30k facce e le regioni generate sono 17, pari al numero di bones (lo skeleton generato da Pinocchio ha sempre questo numero di bones, a meno che non si specifichi un'altro tipo di anatomia non umana).

I tempi delle parti più costose dell'algoritmo (cioè la costruzione del campo di distanza e l'utilizzo della superficie mediale da esso generata per generare il partizionamento dei vertici) sono sempre di una manciata di secondi e sono riportati nella tabella di seguito:

Processore	RAM	Sistema Operativo	Tempo medio
Intel Core 2 Quad Q8200 @ 2.33Ghz	8GB	Windows 7 Professional 64bit	5-7 sec.
Intel Core2 Duo CPU P7350 @2.00Ghz	4GB	Windows Vista Home Premium (SP2) 32 bit	5-7 sec
Intel Core i5-3317U CPU @ 1.70GHz	4GB	Windows 8 64bit	7-14 sec.
Intel Pentium M @ 1.73GHz	1GB	Windows XP Home Edition (SP3)	12-16 sec.

Figura 5.2: *tabella dei tempi di esecuzione della parte più lenta del nostro algoritmo.*

Tutte le altre fasi dell'algoritmo hanno tempi estremamente più bassi, di ordine inferiore ai secondi. In particolare il tempo di calcolo delle distanze geodetiche a partire da ogni bordo di ogni regione è inferiore al secondo, anche sul computer più lento su cui abbiamo testato l'algoritmo.

5.3 Assegnazione dei pesi coerente con la geometria

Grazie all'utilizzo della distanza geodetica che si muove sulla superficie si superano tutti i problemi di assegnazione incoerente dei pesi che abbiamo esaminato nel paragrafo 2.1.2 . Poiché ad ogni regione corrisponde un bone, regioni e bones sono topologicamente equivalenti. Muovendoci con la distanza geodetica partendo dalle regioni garantiamo che l'ordine di valutazione dei vertici segua non solo il loro ordinamento sulla superficie, ma anche la precedenza nella gerarchia di bones. Un vertice di una regione viene raggiunto a partire dalle regioni adiacenti. Così si risolvono i casi in cui per un vertice viene dato troppo peso a un bone cui non appartiene anatomicamente.

Per quanto riguarda le aree di massima deformazione, abbiamo verificato che, per le mesh testate, le fasce in cui le deformazioni si manifestano maggiormente rispettano l'innesto dello skeleton nella mesh. In altre parole tanto più i joints si trovano nelle articolazioni anatomicamente più adatte, tanto più le fasce di

deformazione sono anatomicamente quelle che ci aspetteremmo. Questo perché il rigging non influenza solo i tempi computazionali, ma il suo risultato determina anche come la mesh viene decomposta. In particolare determina i bordi della segmentazione, che, come abbiamo detto, alla fine corrisponderanno alle aree di maggior deformazione della mesh.

In caso il joint sia messo in una posizione poco adatta la fascia di massima deformazione appare in una zona poco adatta. Questo difetto dipende ovviamente dal rigging di Pinocchio e non dagli algoritmi di calcolo dei pesi.

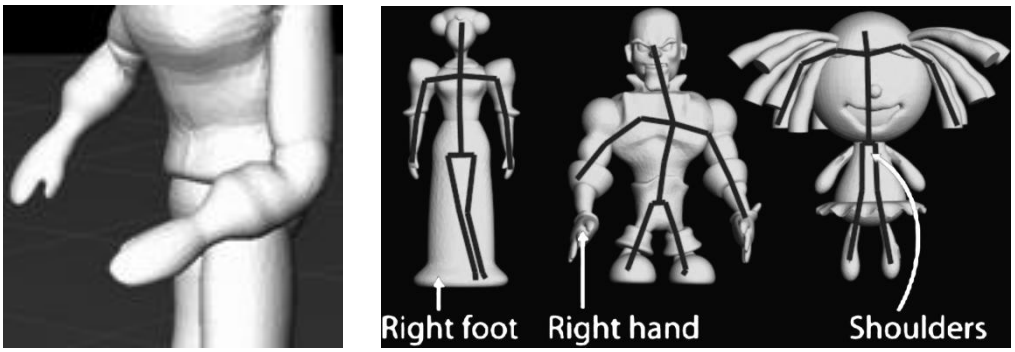


Figura 5.3: *nella figura a sinistra il braccio si piega in un punto dell'avambraccio invece che sulla zona che è visivamente più facile immaginare come gomito; questo accade sia in Pinocchio che nel nostro algoritmo, perché il rigging su cui entrambi si basano ha inserito il joint proprio sull'avambraccio. Nella figura a destra sono mostrati alcuni casi in cui il rigging di Pinocchio fallisce.*

Comunque, nei casi in cui il rigging da un buon risultato, le fasce di massima deformazione del nostro algoritmo sono anatomicamente ottimali e, come vedremo, sono più coerenti di quelle di Pinocchio. Questo proprio perché il nostro algoritmo parte dall'anatomia della mesh e non da quella dello skeleton.

5.4 Riduzione degli artefatti

Così come nell'algoritmo Atlas-based originale ed in Pinocchio, il nostro algoritmo non può evitare gli artefatti di perdita di volume insiti nell'LBS stesso: il collapsing elbow e il candy-wrapper. Questo perché non cambia la natura lineare della

combinazione del peso; è comunque integrabile con una qualsiasi delle evoluzioni basate su LBS, che mirano a ridurre gli artefatti e di fatto nella nostra demo utilizziamo quaternioni per ridurre i difetti.

Tuttavia il nuovo metodo di calcolo dei pesi fornisce una riduzione maggiore degli artefatti di piegamento rispetto al secondo metodo descritto nel paragrafo 3.5.3 che avevamo implementato in principio. Questi miglioramenti sono evidenti per zone come l'inguine: in questa regione si crea comunque una certa convessità tipica del collapsing elbow, ma di dimensione molto minore rispetto al secondo metodo. Osserviamo che provando a mantenere costante il $\sigma_j(v)$ del main-bone e normalizzando tutti gli altri si elimina totalmente il difetto, ma purtroppo si possono generare alcuni artefatti.

Figura 5.4: *risultati per la zone inguinale. Le immagini a partire da sinistra descrivono: i risultati con il secondo metodo per il calcolo dei pesi; i risultati con il nuovo metodo che abbiamo definito; i risultati se manteniamo costante il peso del main-bone; gli artefatti che si generano mantenendo costante il peso del main-bone.*

Tuttavia grazie a questa osservazione possiamo capire che questo tipo di artefatto si genera perché i bones dell'inguine perdono troppa influenza sui vertici delle cosce prossimi all'inguine stessa. Ciò è dovuto alla normalizzazione finale:

$$w_j(v) = \frac{\sigma_j(v)}{\sum_{i \in I(v)} \sigma_i(v)}$$

Con questa formula infatti al main-bone viene sottratto peso da più di un bone quanto più ci si avvicina al bordo e la cosa si ripercuote maggiormente quando più bordi sono molto vicini, come nel caso dei vertici della coscia vicino all'inguine.

In questo caso è Pinocchio ad avere un comportamento migliore rispetto al nostro algoritmo, eliminando totalmente il problema anche se poi vedremo che ne genererà altri per movimenti ampi.

5.5 Problemi di uniformità della mesh

Nonostante l'assegnazione dei pesi sia coerente e lo sia anche la loro espansione su vertici adiacenti, il problema maggiore di questo metodo è che si presentano alcune imperfezioni di uniformità. Le superfici non appaiono sempre lisce e uniformi in tutte le parti della mesh durante l'animazione. Questo si verifica soprattutto quando le torsioni attorno ad un joint che connette più bones diventano estreme. La causa principale probabilmente deriva dall'utilizzo di un'implementazione classica di Dijkstra. Infatti nonostante l'ordine di visita dei nodi sia comunque coerente con la definizione di distanza geodetica, le distanze non sono calcolate nella maniera più adatta. Il fronte d'onda non dovrebbe essere composto da un insieme di punti, ma da un'insieme di segmenti e le distanze di aggiornamento dei vertici dovrebbero essere calcolate come distanze tra un vertice ed un edge, non tra due vertici (si veda APPENDICE C). Dunque le distanze finali calcolate su cui poi si calcolano i pesi non sono le migliori nonostante siano coerenti. Un'altra possibile causa è l'utilizzo del raggio per stimare la distanza dai bordi della regione estesa e dalla parte rigida della mesh, invece di esplicitare questi due limiti e calcolare direttamente le loro distanze.

5.6 Confronto con Pinocchio

Abbiamo già fatto diversi paragoni con i risultati offerti da Pinocchio, ma vediamoli più nel dettaglio, mostrandone anche di nuovi ed analizzando prima due differenze che dipendono dalle implementazioni (mesh trattabili e costi

computazionali) e poi due che dipendono dai principi teorici su cui si fondano (fasce di massima deformazione ed artefatti).

5.6.1 Mesh trattabili

Una differenza che dipende da come è stato implementato il nostro metodo rispetto a quello di Pinocchio sta nelle mesh trattabili. Infatti come abbiamo visto, poiché nel nostro metodo usiamo Dijkstra classico, non abbiamo bisogno che le mesh siano manifold o senza bordi, tantomeno che siano completamente connesse: basta che sia completamente connessa solo la parte che contiene lo skeleton, mentre il resto verrà trattato come corpo rigido. Al contrario Pinocchio richiede che le mesh siano uniche, completamente chiuse e manifold.

5.6.2 Costi

Pinocchio applica il suo metodo di calcolo dei pesi tramite il costruttore di un oggetto `Attachment`, che conterrà appunto i pesi dei bones per ogni vertice. Per generare tale oggetto al suo costruttore vanno passati la mesh, lo skeleton con i suo embedding nella mesh ed infine un oggetto di tipo `VisTester<TreeType>`. Il costruttore di un oggetto `VisTester<TreeType>` necessita a sua volta di un campo di distanza, generato ancora tramite il metodo `constructDistanceField`. Ciò significa che, anche quando è garantito che lo skeleton sia già perfettamente posizionato all'interno della mesh, Pinocchio necessita comunque del campo di distanza per determinare i pesi. Questo è anche il collo di bottiglia del nostro metodo, dunque i costi computazionali sono praticamente equivalenti.

5.6.3 Fasce di massima deformazione

Abbiamo detto come il nostro algoritmo individua meglio le fasce di massima deformazione rispetto a Pinocchio. Ciò accade perché, come abbiamo visto nel paragrafo 2.3 , Pinocchio e più in generale i metodi di diffusione del calore tengono in considerazione l'anatomia dello skeleton invece di quella della mesh, partendo infatti proprio dai bones per diffondere il calore. In altre parole questi metodi si rifanno in modo implicito all'utilizzo dei piani di bisezione e della distanza Euclidea. Riesce comunque a superare i problemi di assegnazione coerente dei pesi rispetto alla geometria, presenti invece nei metodi basati su distanza Euclidea, grazie al fatto di muoversi tramite il volume, ma individua le fasce di deformazione più appropriate solamente quando la mesh presenta un'anatomia che ricalca molto l'anatomia dello skeleton (per esempio sulle joints di gomiti e ginocchia). Al contrario il nostro algoritmo sfrutta l'anatomia della mesh per generare bordi che poi diventeranno le aree di massima influenza ed in questo modo trova appunto fasce di massima deformazione più coerenti. Questa cosa ci appare chiara se osserviamo ad esempio quale è il bone che ha maggior peso su ogni vertice nei due algoritmi.

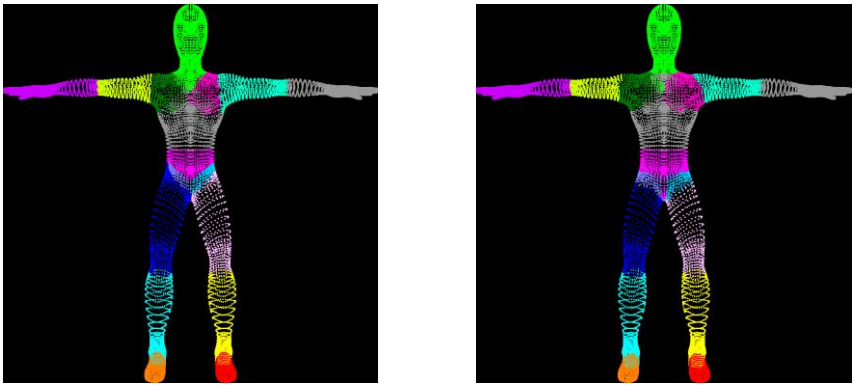


Figura 5.5: *nelle immagini i vertici sono colorati con un colore diverso a seconda del bone che ha maggiore influenza su di essi. Questo significa che le zone di massima deformazione coincidono con i confini dove il colore cambia. L'immagine di sinistra si riferisce ai risultati di Pinocchio. Osserviamo come ad esempio i vertici sui fianchi siano maggiormente influenzati dai bones delle cosce, mentre sarebbe anatomicamente più coerente che fossero maggiormente influenzati dai bones delle anche, cosa che avviene invece con il nostro algoritmo, come possiamo osservare nell'immagine a destra. Si osservi anche la regione del collo e la spalla sinistra. Si noti inoltre come l'immagine di destra sia quasi perfettamente identica alla* Figura 4.12, *che mostra i main-bones per ogni vertice: ciò dimostra come i bordi vengono realmente trasformati in fasce di massima deformazione.*

5.6.4 Artefatti

Dal punto di vista del risultato Pinocchio genera delle superfici sempre molto morbide ed uniformi, al contrario del nostro metodo che mostra risultati a volte discontinui per flessioni estreme degli arti. Quindi Pinocchio dà risultati visivamente più armoniosi del nostro algoritmo. D'altro canto, anche tralasciando la questione dell'individuazione delle fasce di massima deformazione, Pinocchio fornisce dei risultati anatomici meno corretti. Come abbiamo detto nel paragrafo 1.5.1 la perdita di volume è dovuta al fatto che si usa una combinazione lineare rigida dei bones coinvolti e tanti più bones sono coinvolti, tanto più la perdita di volume sarà accentuata. Ad esempio per tutti i vertici interni che si trovano nel tronco e nel bacino, Pinocchio fornisce quasi sempre un uso effettivo di molte bones. Questo si traduce in una maggiore perdita di volume, specialmente quando tanti bones incidenti in un unico joint vengono mossi assieme. Il nostro algoritmo invece limita l'estensione di utilizzo di un bone grazie proprio alla determinazione dell'area di estensione. Per esempio abbiamo quando il personaggio è in piedi, come abbiamo visto, il nostro metodo genera un artefatto di concavità nella zona inguinale che in Pinocchio invece non c'è. Tuttavia Pinocchio presenta un altro genere di artefatto nella stessa zona nel momento in cui le gambe si divaricano maggiormente. Quando ciò avviene la zona inguinale si appiattisce eccessivamente, alterando quindi l'anatomia della figura.

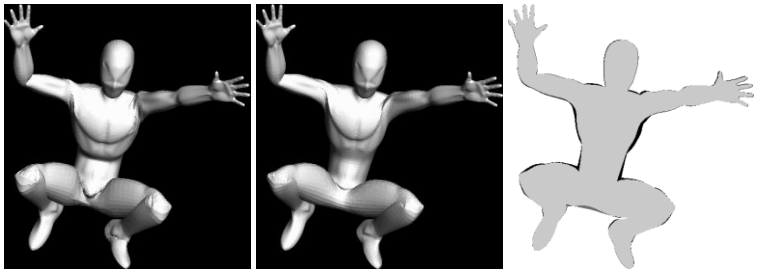


Figura 5.6: *prendiamo qui una posizione estrema (di quelle in cui il nostro algoritmo dà il peggio) e osserviamo le immagini a partire da sinistra: la prima generata con il nostro metodo presenta diversi artefatti di discontinuità proprio perché la posa è estrema, tuttavia il volume della mesh sul tronco del corpo è maggiormente preservato così come la zona inguinale; la seconda immagine è generata tramite Pinocchio e mostra una superficie più uniforme, tuttavia il torace presenta una perdita di volume maggiore e la zona inguinale si è totalmente appiattita; nell'ultima immagine sono sovrapposte le sagome delle prime due figure per evidenziare la perdita di volume. In particolare si osservi nelle tre figure la diversa volumetria della zona toracica.*

5.7 Confronto con l'algoritmo originale

L'algoritmo mantiene tutti i vantaggi sia del metodo generale che della sua generalizzazione:

- preserva la sua intuitività e semplicità. La decomposizione si basa su metodi di rigging i quali preservano intrinsecamente la relazione tra la geometria e lo skeleton; i pesi sono definiti tramite una formula semplice ed indipendente tra i vertici,
- il calcolo dei pesi è totalmente automatizzato e non necessita di inserimento di parametri,
- è integrabile con l'LBS standard o con una delle sue evoluzioni,
- in input ha bisogno solo dello skeleton e della mesh.
- è veloce tanto quanto il metodo di rigging su cui si basa la decomposizione.

Questi stessi vantaggi sono inoltre espandibili. Infatti, se lo si desidera, il metodo può comunque essere parametrizzato su ogni regione tramite la modifica dei bordi e/o della distanza di estensione, oppure selezionando uno tra diversi metodi di decomposizione basati su rigging, o anche scegliendo uno tra i diversi tipi di calcolo della distanza (sia tra diversi tipi di distanza geodetica che tra tipi di altro genere). Potremmo così decidere se avere un metodo completamente automatizzato o semi-automatizzato.

Questo vantaggio viene dalla modularità intrinseca dell'algoritmo. Le tre fasi principali sono indipendenti l'una dall'altra e possono essere viste in pipeline. Ogni fase può essere sostituita con un metodo alternativo. Abbiamo visto come per decomporre la mesh esistano una gran quantità di algoritmi. Alla stessa maniera si può usare una qualsiasi tipologia di distanza o anche una composizione di distanze (ricordiamo come nella nostra implementazione venga usata la distanza Euclidea per

trattare parti sconnesse in modo rigido) per calcolare le aree di sovrapposizione e/o per il calcolo dei pesi.

Inoltre è possibile modificare ulteriormente il metodo per generare automaticamente uno skeleton (invece di riceverlo in input) adatto ad una qualsiasi anatomia, anche non umana. Questa modifica vien proprio grazie al fatto che la decomposizione è originariamente pensata per basarsi su metodi che generano skeleton: ricordiamo difatti che sfruttiamo metodi di rigging con gli stessi principi dei metodi che generano skeleton.

Non bastasse, a questi vantaggi il nostro metodo ne aggiunge diversi altri, non presenti nell'algoritmo originale. Come prima cosa durante la decomposizione non si accorpano i bones se più di 2 incidono su un unico joint. Ciò significa che non perdiamo i movimenti descritti dalle rotazioni centrate su tale joint.

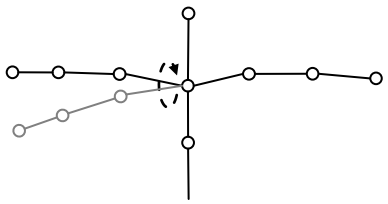


Figura 5.7: *le rotazioni della clavicola non sono descrivibili nelle due versioni originali dell'algoritmo, mentre lo sono con il nostro algoritmo.*

Altro vantaggio è che le fasce di massima deformazione non sono più descritte tramite un unico piano di taglio e quindi i bordi rispecchiano meglio l'anatomia della mesh (si veda più sopra il paragrafo 3.6.1 Analisi della decomposizione).

Oltre a ciò la distanza di estensione da un bordo può non essere la stessa nelle 2 direzioni: quella verso l'interno della regione e quella verso l'esterno. Difatti nel nostro algoritmo dato un bordo noi calcoliamo tale distanza diversamente a seconda della regione considerata, rispetto cioè al bone della regione (anche se si può sempre scegliere di rendere questi due valori identici). Dunque volendo, non solo si possono provare diversi modi per scegliere le distanze di estensione, ma si possono provare

anche distanze diverse a seconda se ci si muove verso l'interno o verso l'esterno della regione.

Infine, per quanto concerne il calcolo dei pesi, usando il diametro come valore per uniformare i pesi (come si faceva nel primo metodo, invece che utilizzare δ come nel secondo metodo) e muovendoci dai bordi, evitiamo che le aree di massima deformazione si discostino troppo dai bordi stessi trovati con la decomposizione.

Di contro, per movimenti ampi su joints in cui incidono più di 2 bones, l'utilizzo di una distanza geodetica calcolata con il semplice Dijkstra, anziché con metodi più raffinati genera a volte superfici poco uniformi e artefatti. È quindi consigliabile testare uno dei metodi di calcolo della distanza geodetica più evoluto.

Conclusioni e sviluppi futuri

Questa tesi affrontava il problema dell'individuazione di un metodo di skinning automatico per lo sviluppo di un'applicazione di animazione real-time per Virtual Humans. A tale scopo in principio abbiamo individuato il più diffuso metodo di animazione per software in tempo reale quale l'LBS. Abbiamo quindi analizzato e confrontato alcuni algoritmi per il calcolo dei pesi utilizzati nello skinning, presentando, tramite questi, le problematiche più note e caratteristiche. Tra questi algoritmi si è quindi scelto di implementare un metodo che sfrutta la distanza geodetica, in grado di risolvere gran parte dei maggiori problemi oltre ad essere adattabile alla trattazione di mesh poco pulite acquisite da strumenti di scansione tridimensionale. Abbiamo quindi descritto l'algoritmo nelle sue diverse fasi di decomposizione, estensione delle regioni e calcolo dei pesi, assieme ad una sua generalizzazione più recente. Siamo poi passati ad un'analisi delle fasi più interessanti dell'algoritmo e da quest'analisi abbiamo individuato alcune problematiche. Sulla base di queste problematiche abbiamo sviluppato e descritto delle nuove varianti delle fasi, mirate a risolvere alcuni difetti individuati. Abbiamo poi implementato l'intero algoritmo rinnovato in un tool che ci mostrasse visivamente i risultati.

Il metodo si è mostrato semplice ed efficiente, ma soprattutto con grandi possibilità di sviluppo. Risolve gran parte dei problemi presenti nella distanza

Euclidea. Rispetto al metodo di diffusione del calore tiene maggiormente conto della geometria, proprio grazie alla decomposizione iniziale.

Riguardo i possibili sviluppi futuri, dal punto di vista implementativo si può pensare di integrare i valori di parametrizzazione discussi nel paragrafo 5.7 (scelta del metodo di decomposizione, modifica dei bordi, scelta della distanza di estensione nelle 2 direzioni, scelta del tipo di distanza) così da comparare i risultati e adattarli alle più diverse esigenze. In particolare, usando la distanza armonica, si potrebbero esaminare risultati simili a quelli del metodo di Pinocchio. Tale metodo, come spiegato nel paragrafo 2.3.1, sfrutta una funzione di equilibrio del calore per stabilire i pesi, ottenendo dei risultati molto morbidi, ma individuando aree di massima deformazione meno precise. Usandolo assieme al nostro metodo si potrebbero individuare meglio tali aree.

Per quanto riguarda l'algoritmo sviluppato, basato su distanza geodetica (paragrafo 4.3.1), una linea di sviluppo potrebbe essere quella di testare altri metodi di calcolo di tale distanza più raffinati, che generino meno artefatti per i movimenti estremi degli arti. Questi algoritmi devono comunque essere adattati alla trattazione di mesh sporche (aperte e/o non manifold). In alternativa si può tentare di implementare una pre-elaborazione della mesh che la ripulisca in modo anatomicamente coerente.

Nel calcolo delle $\sigma_j(v)$ (nel paragrafo 4.4) si può provare a sostituire la distanza $D_{i,k}$ (ottenuta come illustrato nel paragrafo 4.3.2) con una distanza più precisa, esplicitando i bordi interni ed esterni e calcolando la distanza da questi. Questo comporterebbe dei calcoli aggiuntivi, ma sarebbe interessante testare gli eventuali conseguenti benefici.

Appare promettente l'implementazione di un metodo di decomposizione e (di conseguenza) di rigging più preciso di quello usato da Pinocchio. Un metodo che potrebbe essere molto interessante, per la sua capacità di portarsi dietro la geometria, è quello basato sui diagrammi di Voronoi (di cui abbiamo visto più volte come

descrivano in modo naturale e preciso la suddivisione dello spazio) ma partendo dalle facce, non dallo skeleton e sfruttando il risultato per raggiungere dai vertici i bones. Esistono diversi algoritmi per generare i diagrammi di Voronoi in uno spazio 3D, come ad esempio quello di Bowyer-Watson [(Watson, 1981), (Bowyer, 1981)]. Al di là di questo specifico algoritmo, sarebbe interessante testare gli altri metodi citati in APPENDICE B.

Concludendo, al fine di consentire una più completa e obiettiva valutazione di questi algoritmi, il tool potrebbe essere espanso per implementare le metriche di valutazione trattate nel paragrafo 5.1 (Jacka, et al., 2007).

APPENDICE A

LBS con quaternioni duali

Ricalcando uno studio esistente (Zhai, et al., 2011) spieghiamo come mai le perdite di volume nell'LBS siano maggiori per grandi angoli di rotazione mostrando le matrici di trasformazione per due bones. Come nel paragrafo 1.5.1 supponiamo di avere solo rotazioni attorno all'asse y e teniamo fisso il bone $j = 1$, ovvero la trasformazione applicata da tale bone è l'identità: $C_1 = I_1$. Avremo quindi:

$$v = \sum_{j=1}^{2} w_j C_j \hat{v} = (w_1 I_1 + w_2 C_2)\hat{v} = \left(w_1 I_1 + w_2 \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right) \hat{v}$$

$$= \left(\begin{bmatrix} w_1 + w_2\cos\theta & 0 & -w_2\sin\theta & 0 \\ 0 & w_1 + w_2 & 0 & 0 \\ w_2\sin\theta & 0 & w_1 + w_2\cos\theta & 0 \\ 0 & 0 & 0 & w_1 + w_2 \end{bmatrix} \right) \hat{v}$$

$$= \begin{bmatrix} (w_1 + w_2\cos\theta)x - (w_2\sin\theta)z \\ (w_1 + w_2)y \\ (w_2\sin\theta)x + (w_1 + w_2\cos\theta)z \\ 1 \end{bmatrix}$$

Il motivo per cui la distorsione è maggiore per angoli grandi è che, come vediamo, le coordinate finali del vertice di trasformazione v contengono oggetti coseno. Infatti quando $\theta > 90°$, allora $\cos\theta < 0$ e quindi le coordinate del vertice

diminuiranno. Inoltre più θ è prossimo a 180° più le coordinate avranno una riduzione di scarto maggiore.

Il problema è che non si può garantire l'ortogonalità matriciale. Possiamo però usare quaternioni che sono strumento molto adatto a descrivere rotazioni e sono numericamente più stabili dell'uso di matrici di rotazione.

Matematicamente un quaternione q è un elemento del tipo:

$$q = s + xi + yj + zk$$

con s, x, y, z reali, mentre i, j, k letterali, per i quali vale $i^2 = j^2 = k^2 = ijk = -1$. La parte immaginaria $xi + yj + zk$ è rappresentabile da un vettore v nello spazio tridimensionale, mentre s è uno scalare reale. Poiché i, j, k sono costanti per qualsiasi quaternione, possiamo rappresentare un generico quaternione q come un vettore di 4 reali (s, x, y e z) o anche come un reale s ed un vettore v:

$$q = [s, x, y, z] = [s, v] \quad \text{con} \quad v = (x, y, z) \in \mathrm{R}^3 \quad \text{e} \quad s \in \mathrm{R}^3$$

Definiamo il coniugato di q con $\bar{q}$, la norma di q con $\|q\|$ e l'inversa di q con q^{-1}:

$$\bar{q} = [s, -v] \qquad \|q\| = \sqrt{s^2 + x^2 + y^2 + z^2} \qquad q^{-1} = q/\|q\|$$

La moltiplicazione tra due quaternioni $q_1 = [s_1, v_1]$ e $q_2 = [s_2, v_2]$ si può esprimere tramite il normale prodotto scalare e quello vettoriale (a causa di quest'ultimo non è commutativa):

$$q_1 q_2 = q_1 \times q_2 - q_1 \cdot q_2 = [s_1 s_2 - v_1 v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2]$$

Se $\|q\| = 1$ allora q è detto quaternione unitario. Se due quaternioni p e q sono unitari e l'angolo di inclinazione è θ, allora $(p, q) = \cos\theta$.

Usiamo quaternioni nella forma $q = \left[\cos\frac{\theta}{2}, n\sin\frac{\theta}{2}\right]$ per esprimere rotazioni. Se vogliamo ruotare un vettore unitario r di un angolo θ attorno all'asse n, il vettore unitario ruotato r' sarà calcolato tramite $[0, r'] = q[0, r]q^{-1}$. Questa è la forma con la quale i quaternioni esprimono le rotazioni. Allo stesso tempo si può dimostrare che q e $-q$ esprimono la stessa rotazione.

Se espandiamo l'equazione base dell'LBS come abbiamo fatto prima, ma stavolta usando quaternioni invece di matrici di trasformazione, otteniamo:

$$v = \sum_{j=1}^{2} w_j C_j \hat{v} = w_1 I_1 + w_2 q[0, r]q^{-1} =$$

$$= w_1 I_1 + w_2 \left[0, \left(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}\right) r + 2\left(\sin^2\frac{\theta}{2}\right) n(n - r) + (\sin\theta) n \times r\right] =$$

$$= w_1 I_1 + w_2 [0, (\cos\theta) r + (1 - \cos\theta) n(n - r) + (\sin\theta) n \times r] =$$

$$= \begin{bmatrix} (w_1 + w_2\cos\theta)x - (w_2\sin\theta)z \\ (w_1 + w_2)y \\ (w_2\sin\theta)x + (w_1 + w_2\cos\theta)z \\ 1 \end{bmatrix}$$

Come prima abbiamo preso come asse di rotazione l'asse delle *y*: $n = [0,1,0]$.

Osserviamo intanto che se $\theta < 90°$, il risultato è lo stesso dell'equazione LBS con matrici. Invece quando $\theta > 90°$ e di conseguenza $\cos\theta < 0$, conviene utilizzare il quaternione $-q$ al posto di q. Questo è possibile perché q e $-q$ definiscono la stessa rotazione. Dato che in questo caso abbiamo $(p, -q) = \cos\theta = \cos(\pi + \theta) > 0$, possiamo lasciare l'utilizzo di θ dove si usa la funzione seno, mentre laddove si usa il coseno possiamo utilizzare $\pi + \theta$. Così il calcolo della nuova posizione del vertice diventa:

$$v = \sum_{j=1}^{2} w_j C_j \hat{v} = \begin{bmatrix} (w_1 + w_2 \cos(\pi + \theta))x - (w_2 \sin\theta)z \\ (w_1 + w_2)y \\ (w_2 \sin\theta)x + (w_1 + w_2 \cos(\pi + \theta))z \\ 1 \end{bmatrix}$$

Se $\theta > 90°$, stavolta sulle parti dove appare il coseno avremo che $\cos(\pi + \theta) > 0$ e quindi si riesce ad evitare la distorsione.

Un altro grande vantaggio è che si possono sempre trasformare matrici in quaternioni.

APPENDICE B

La decomposizione di superfici

La decomposizione di una mesh in parti significative può essere trattata come un problema di ottimizzazione non semplice da risolvere automaticamente, anche se esistono diversi metodi [(Liu, et al., 2007) e (Krayevoy, et al., 2006)]. In particolare, come indicato anche in "Atlas-Based Character Skinning with Automatic" (Lu, et al., 2008), la decomposizione di superfici è un problema ampiamente trattato in letteratura per moltissimi motivi. Di seguito ne citiamo alcuni:

- la semplificazione ed il remeshing [(Zuckerberger, et al., 2002) e (Cohen-Steiner, et al., 2004)]
- la parametrizzazione [(Krayevoy, et al., 2004) e (Sander, et al., 2001)]
- l'analisi o modellazione della forma [(Krayevoy, et al., 2007) e (Mortara, et al., 2004)]
- la modifica o la riparazione della mesh (Lee, et al., 2005)
- il matching della forma (Zuckerberger, et al., 2002)
- l'estrazione dello skeleton [(Katz, et al., 2003), (Lien, et al., 2006) e (Pratscher, et al., 2005)]
- il morphing e la metamorfosi [(Shlafman, et al., 2002) e (Zuckerberger, et al., 2002)]
- il rilevamento di collisioni [(Li, et al., 2001) e (Lu, et al., 2007)]

Nello sviluppare l'algoritmo Atlas-based che abbiamo studiato, gli autori hanno optato per un approccio in cui si suppone che lo skeleton sia dato come input e che sia già interno alla mesh. La decomposizione avviene proprio sfruttando tale skeleton. Un approccio simile è usato in "Outside-in anatomy based character rigging" (Pratscher, et al., 2005), in cui si segmenta la mesh basandosi sulla vicinanza ai bones dello skeleton, e poi si applicano alcuni test per correggere i casi in cui l'assegnamento di vicinanza di un vertice fallisce. L'algoritmo che abbiamo studiato invece si concentra sui joints dello skeleton. Poiché l'algoritmo mira ad associare le regioni della mesh alle articolazioni, conviene puntare direttamente alla segmentazione ricercando i confini.

Una classificazione degli algoritmi di decomposizione in diverse categorie è descritta in "Segmentation and Shape Extraction of 3D Boundary Meshes" (Shamir, 2006). Questi sono metodi basati su aumento di regione, spartiacque, clustering, metodi di analisi spettrale, taglio-di-grafo, metodi impliciti, inferiti da uno skeleton. Il metodo sfruttato dall'algoritmo che abbiamo studiato appartiene ad entrambe le ultime due categorie: è un approccio implicito basato-su-skeleton; questo perché si calcolano le regioni rilevando i loro confini, come trattato in "Mesh Scissoring with Minimal Rule and Part Salience" (Lee, et al., 2005), ma si utilizza uno skeleton per aiutare il processo di decomposizione, come avviene in altri studi [(Lu, et al., 2007) e (Pratscher, et al., 2005)].

APPENDICE C

Calcolo delle distanze geodetiche

La distanza geodetica descrive localmente la traiettoria più breve fra 2 punti muovendosi al di sopra di una superficie. Esistono molti metodi per calcolare la distanza geodetica tra due punti. Questi si basano tutti su varianti dell'algoritmo di Dijkstra. Ricordiamo che tale algoritmo calcola i cammini minimi (e quindi anche la distanza minima) in un grafo orientato o meno, anche ciclico, con pesi sugli archi non negativi, partendo da uno o più nodi sorgente e (sfruttando apposite strutture) in tempo lineare nel numero di archi e vertici (il costo è pari a $O(|E| + |V| \log |V|)$, dove $|E|$ è il numero di archi e $|V|$ il numero di nodi).

Abbiamo già detto che in effetti una mesh "potrebbe" essere vista come un grafo non orientato, ciclico se: come nodi prendiamo i vertici, come archi orientati prendiamo gli half-edge e come peso degli archi potremmo ad esempio prendere la distanza Euclidea tra i due vertici estremi di ogni edge (che non sarà mai negativa, così come vuole l'algoritmo).

Ricordiamo anche che, in ogni istante, l'algoritmo di Dijkstra mantiene i nodi divisi in 3 partizioni o insiemi (*set*): un insieme di nodi ancora da esaminare (*Unprocessed vertex*), un'altro di nodi per cui la distanza minima è stata trovata e fissata (*Fixed vertex*), ed un ultimo insieme che tiene i vertici per cui la distanza

minima potrebbe essere ancora migliorata e che costituisce quindi un fronte d'onda (*Close vertex*).

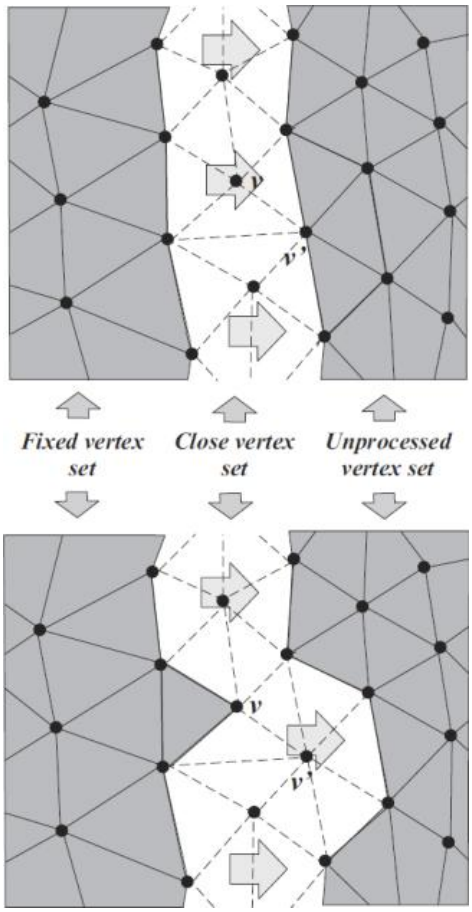


Figura C.0.1: *i tre set dell'algoritmo di Dijkstra. Il fronte d'onda (Close vertex) si sposta.*

Brevemente, l'algoritmo si basa su un ciclo in cui:

- si estrae da Close il vertice *v* che ha valore di distanza minimo e viene messo in Fixed (stabilendo così che ormai per *v* abbiamo trovato la più piccola distanza possibile)
- si attraversano tutti gli archi che vanno da *v* ad un altro vertice *v'* contenuto in Close ∪ Unprocessed
    - se *v'* era in Unprocessed viene spostato in Close

- se $d(v') > d(v) + \text{length}(v,v')$ allora $d(v') = d(v) + d(v,v')$, dove length(v,v') è la lunghezza dell'arco da v a v' [*Fase di aggiornamento*]

- si ricomincia il ciclo finché ci sono vertici in Close.

La differenza tra l'algoritmo classico e i metodi di calcolo della distanza geodetica sta tutta nella fase di aggiornamento.

In un'implementazione standard di Dijkstra il valore di distanza di un vertice v' che appartiene a Close o a Unprocessed, viene aggiornato quando si valuta un suo vertice adiacente v che è nel fronte d'onda (in Close) ed è stato appena spostato in Fixed. Si calcola quindi una nuova distanza (candidata) per v' sfruttando il peso dell'arco (v,v'). Se la vecchia distanza che aveva v' è maggiore della nuova, allora la vecchia viene sostituita. Sottolineiamo come l'arco considerato abbia un vertice estremo nella parte fissa (v è in Fixed) e un vertice nella parte che può ancora cambiare (v' è in Close o in Unprocessed).

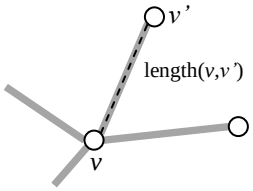


Figura C.0.2: *v è il vertice che passa da Close a Fixed e v' è un vertice in Close o in Unprocessed. La linea tratteggiata indica la lunghezza dell'arco usata per un'eventuale aggiornamento della distanza di v' e passa sempre per un edge.*

Nell'implementazione per calcolare la distanza geodetica non viene usato per forza un'edge tra due vertici. Questo perché essendo la mesh la rappresentazione di una superficie (e non di un grafo, come invece semmai lo sono i suoi edges), il vertice può essere raggiungibile anche passando sopra la superficie delle facce (camminando sopra i triangoli insomma) e non solo attraversando gli edges. Dunque in questi metodi il peso viene invece calcolato tra un edge e ed un vertice v' (praticamente tra un edge ed il suo vertice contrapposto su di una faccia). In particolare l'edge e ha entrambi i vertici in Fixed, uno dei quali è il vertice a distanza minima che spostiamo

da Close a Fixed (in pratica è il *v* dell'implementazione classica), mentre il vertice *v'* appartiene a Close o a Unprocessed (come prima).

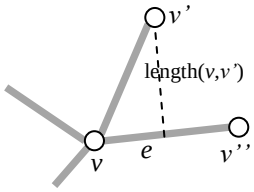


Figura C.0.3: *v è il vertice che passa da Close a Fixed, v'' è un altro vertice in Fixed e v' è un vertice in Close o in Unprocessed. La linea tratteggiata indica la lunghezza usata per un'eventuale aggiornamento della distanza di v' e che, come vediamo, non passa per forza da un arco.*

In un certo senso si può dire che il fronte d'onda in questi algoritmi non è costituito da punti, ma da edges, i cui estremi sono vertici in Fixed e che entrambi tali estremi hanno almeno un vertice adiacente non in Fixed. Gli algoritmi di distanza geodetica si basano su questo principio raffinando in diversi modi il calcolo della distanza di aggiornamento tra l'edge e il vertice.

Possiamo anche dire che Dijkstra classico è un'approssimazione "debole" della distanza geodetica: come se, per calcolare la distanza tra due punti di una superficie, si potesse passare solo tramite strade predefinite (gli edge), anziché su qualsiasi punto della superficie (sopra le facce).

BIBLIOGRAFIA

Alexa Marc Linear combination of transformations [Rivista] // SIGGRAPH: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques. - 2002. - p. 380–387.

Aujay Grégoire [et al.] Harmonic Skeleton for Realistic Character Animation [Rivista] // Symposium on Computer Animation. - 2007.

Autodesk Maya Smooth Bind [Articolo] // http://download.autodesk.com/global/docs/maya2014/en_us/index.html?url=files/Skin__Bind_Skin__Smooth_Bind.htm,topicNumber=d30e325229. - 2014.

Baran Ilya e Popović Jovan Automatic Rigging and Animation of 3D Characters [Rivista] // ACM SIGGRAPH conference proceedings Article No. 72. - 2007.

barnabas79 PM_heatWeight 0.6.6 (maya script) [Articolo] // http://www.creativecrash.com/maya/script/pm_heatweight. - 2010.

BlenderEN Your First Animation in 30 plus 30 Minutes Part II [Articolo] // http://wiki.blender.org/index.php/Doc:2.6/Manual/Your_First_Animation/2.Animating_the_Gingerbread_Man.

BlenderIT La tua prima animazione in 30 + 30 minuti Parte II [Articolo] // http://wiki.blender.org/index.php/Doc:IT/2.6/Manual/Your_First_Animation/2.Animating_the_Gingerbread_Man.

Botsch Mario, Bommes David e Kobbelt Leif Efficient linear system solvers for mesh processing [Rivista] // IMA Conference on the Mathematics of Surfaces. - p. 62-83.

Bowyer Adrian Computing Dirichlet tessellations [Rivista] // The Computer Journal. - 1981. - p. 162–166.

Cal3D Cal3D - 3d character animation library [Articolo] // http://gna.org/projects/cal3d/.

Carrozzino Marcello e Tecchia Franco A flexible framework for wide-spectrum VR development [Rivista] // PRESENCE: Teleoperators and Virtual Environments. - 4 : Vol. 19. - p. 302-312.

Chen Cheng-Hao [et al.] Lattice-based skinning and deformation for real-time skeleton-driven animation [Rivista] // 12th International Conference on Computer-Aided Design and Computer Graphics. - 2011.

Cohen-Steiner David, Alliez Pierre e Desbrun Mathieu Variational shape approximation [Rivista] // ACM Transactions on Graphics (SIGGRAPH proceedings), 23(3). - 2004. - p. 905-914.

Collins Gordon e Hilton Adrian Models for character animation [Rivista] // Software Focus, Vol 2, Issue 2. - 2001. - p. 44– 51.

Dominio Fabio Stima della traiettoria di una mano a partire da dati 3D [Rivista] // http://tesi.cab.unipd.it/35103/1/Fabio_Dominio_Tesi.pdf. - 2010. - p. 47.

Franco Alberto Strategie Risolutive per Matrici Sparse con Tecnologie GPGPU e Multi-core [Rivista]. - Padova : [s.n.], 2009. - p. 14.

FTLK Fast Light Toolkit [Articolo] // http://www.fltk.org/index.php.

Fusiello Andrea Elementi di Algoritmi Geometrici [Rivista].

Gessler Alexander [et al.] Assimp [Articolo] // http://assimp.sourceforge.net/.

Hétroy Franck [et al.] Simple Flexible Skinning Based on Manifold Modeling [Rivista] // International Conference on Computer Graphics Theory and Applications (GRAPP). - 2009.

Jacka David [et al.] A comparison of linear skinning techniques for character animation [Rivista] // AFRIGRAPH '07 Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa. - 2007.

Katz Sagi e Tal Ayellet Hierarchical mesh decomposition using fuzzy clustering and cuts [Rivista] // ACM Transactions on Graphics (SIGGRAPH proceedings), 22(3). - 2003. - p. 954-961.

kogen Autoweight - Automatic skinning tool for 3ds Max [Articolo] // http://www.scriptspot.com/3ds-max/scripts/autoweight-automatic-skinning-tool-for-3ds-max.

Komura Taku Rigging / Skinning [Articolo] // Computer Animation and Visualisation Lecture 4..

Krasner Anna, Xu David e Hoang Victor Mixamo [Articolo] // http://www.mixamo.com/.

Krayevoy Vladislav e Sheffer Alla Cross-parameterization and compatible remeshing of 3D models [Rivista] // ACM Transactions on Graphics (SIGGRAPH proceedings). - 2004.

Krayevoy Vladislav e Sheffer Alla Variational, Meaningful Shape Decomposition [Rivista] // SIGGRAPH Sketches. - 2006.

Krayevoy Vladislav, Julius Dan e Sheffer Alla Modeling with interchangeable parts [Rivista] // The Visual Computer. - 2007.

Kry Paul G., James Doug L. e Pai Dinesh K. EigenSkin: real time large deformation character skinning in hardware [Rivista] // Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. - 2002. - p. 153–159.

Lee Yunjin [et al.] Mesh Scissoring with Minimal Rule and Part Salience [Rivista] // Computer Aided Geometric Design, 22(5). - 2005. - p. 444-465.

Lewis J. P., Cordner Matt e Fong Nickson Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation [Rivista] // SIGGRAPH. - 2000.

Li Xuetao [et al.] Decomposing polygon meshes for interactive applications [Rivista] // Symposium on Interactive 3D Graphics and Games. - 2001. - p. 35-42.

Lien Jyh-Ming, Keyser John e Amato Nancy Marie Simultaneous shape decomposition and skeletonization [Rivista] // Symposium on Solid and Physical Modeling. - 2006. - p. 219-228.

Liu Rong e Zhang Hao Mesh segmentation via spectral embedding and contour analysis [Rivista] // Computer Graphics Forum (Eurographics proceedings), 26(3). - 2007.

Lu Lin [et al.] Atlas-Based Character Skinning with Automatic [Rivista] // inria-00202597, version 4, N° 6406. - 2008.

Lu Lin [et al.] Variational 3D shape segmentation for bounding volume computation [Rivista] // Computer Graphics Forum (Eurographics proceedings), 26(3). - 2007.

Magnenat-Thalmann Nadia [et al.] Modeling of bodies and clothes for virtual environments [Rivista] // Proceedings of the 3rd International Conference on Cyberworlds (CW). - 2004. - p. 201–208.

Magnenat-thalmann Nadia, Laperrire Richard e Thalmann Daniel Joint-Dependent Local Deformations for Hand Animation and Object Grasping [Rivista] // Graphics Interface. - 1988.

Merry Bruce, Marais Patrick e Gain James Animation Space: a truly linear framework [Rivista] // ACM Transactions on Graphics (TOG), Volume 25, Issue 4. - 2003. - p. 1400-1423.

Meyer Mark [et al.] Discrete differential-geometry operators for triangulated [Rivista] // Visualization and Mathematics III. - 2003. - p. 35–57.

Mohr Alex e Gleicher Michael Building efficient, accurate character skins from examples [Rivista] // ACM SIGGRAPH 2003, 22, 3. - 2003. - p. 562-568.

Mohr Alex, Tokheim Luke e Gleicher Michael Direct manipulation of interactive character skins [Rivista] // Proceedings of the Symposium on Interactive 3D Graphics. - 2003. - p. 27–30.

Mortara Michela [et al.] Plumber: a method for a multi-scale decomposition of 3D shapes into tubular primitives and bodies [Rivista] // Symposium on solid modeling and applications. - 2004. - p. 339-344.

Novotni Marcin e Klein Reinhard Computing geodesic distances on triangular [Rivista] // International Conference in Central Europe on Computer Graphics. - 2002.

Pratscher Michael [et al.] Outside-in anatomy based character rigging [Rivista] // Symposium on Computer Animation. - 2005.

Pratscher Michael [et al.] Outside-in Anatomy Based Character Rigging [Rivista] // Symposium on Computer Animation. - 2005.

Praun Emil, Finkelstein Adam e Hoppe Hugues Lapped Textures [Rivista] // SIGGRAPH. - 2000.

Rohmer Damien, Hahmann Stefanie e Cani Marie-Paule Local Volume Preservation for Skinned Characters [Rivista] // Pacific Graphics Volume 27, Number 7. - 2008.

Roselle Steven Maya 2013 Highlight - New Heat Map Skinning [Rivista] // http://area.autodesk.com/blogs/stevenr/maya_2012_highlight_new_heat_map_skinning . - 2012.

Rosen David Volumetric heat diffusion skinning [Rivista] // http://blog.wolfire.com/2009/11/volumetric-heat-diffusion-skinning/. - 2009.

Sander Pedro [et al.] Texture mapping progressive meshes [Rivista] // ACM SIGGRAPH Proceedings. - 2001. - p. 409-416.

Shamir Ariel Segmentation and Shape Extraction of 3D Boundary Meshes [Rivista] // Eurographics State-of-the-Art Report. - 2006.

Shlafman Shymon, Tal Ayellet e Katz Sagi Metamorphosis of Polyhedral Surfaces using Decomposition [Rivista] // Computer Graphics Forum (Eurographics proceedings), 21(3),. - 2002.

Silva Samuel, Madeira Joaquim e Santos Beatriz Sousa PolyMeCo-An integrated environment for polygonal mesh analysis and comparison [Rivista] // IV '05 Proceedings of the Ninth International Conference on Information Visualisation. - 2005. - p. 842-847.

Sloan Peter-Pike J., Rose III Charles F. e Cohen Michael F. Shape by Example [Rivista] // Proceedings of the Symposium on Interactive 3D Graphics. - 2001. - p. 135–143.

super_oko Auto mesh Skinning [Articolo] // http://www.scriptspot.com/3ds-max/scripts/auto-mesh-skinning.

Tang Jie [et al.] Fast Approximate Geodesic Paths on Triangle Mesh [Rivista] // International Journal of Automation and Computing, 4. - 2003.

Toledo Sivan, Chen Doron e Rotkin Vladimir TAUCS: A library of sparse linear solvers, version 2.2 [Rivista] // http://www.tau.ac.il/~stoledo/taucs/. - 2003.

Trammell Kent Quick Rigging With the Rigify Addon [Articolo] // http://cgcookie.com/blender/2013/04/23/quick-rigging-with-the-rigify-addon/.

Wang Cheng [et al.] Automatic Skeleton Generation and Character Skinning [Rivista] // State Key Lab of Virtual Reality and System, Beihang University.. - p. 299-304.

Wang Cheng e Dai Shuling Muscle Pushing Based Skin Deformation On GPU [Rivista]. - 2010.

Wang Xiaohuan Corina e Phillips Cary Multi-Weight enveloping: Least-Squares approximation techniques for skin animation [Rivista] // Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. - 2002. - p. 129-138.

Watson David F. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes [Rivista] // The Computer Journal. - 1981. - p. 167–172.

Zhai Hui, Li Jiyun e Zhai Rui New method to skin deformation driven by dual quaternions based on LBS [Rivista] // International Conference on Electronic & Mechanical Engineering and Information Technology. - 2011. - p. 4432 - 4434.

Zuckerberger Emanoil, Tal Ayellet e Shlafman Shymon Polyhedral surface decomposition with applications [Rivista] // Computer and Graphics, 26(5). - 2002.