ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

INGEGNERIA INFORMATICA

CORSO DI LAUREA MAGISTRALE

TESI DI LAUREA

in

Fondamenti di Computer Graphics M

ADD-ON IN BLENDER PER LA DEFORMAZIONE DI MESH POLIGONALI

CANDIDATO

Caputo Fabio

RELATORE:

Chiar.mo Prof.ssa

Morigi Serena

Anno Accademico 2012/2013

Sessione II

INDICE

PAROLE CHIAVE

Surface-based deformations

Mesh deformations

Discrete Elastica

Python

I. INTRODUZIONE

L'idea da cui nasce questa tesi è quella di introdurre in Blender un Add-on in linguaggio Python che permetta di applicare alcune deformazioni di tipo surface-based a mesh poligonali. Questa tipologia di deformazioni rappresentano l'alternativa alle deformazioni di mesh poligonali tramite rigging (cioè l'aggiunta di uno scheletro per controllare e per animare la mesh) e caging (cioè l'utilizzo di una struttura di controllo di tipo reticolare che propaga la sua deformazione su un oggetto in essa immerso), che di solito sono le prescelte in computer animation e in modellazione. Entrambe le deformazioni indicate sono già estremamente radicate in Blender, prova ne è il fatto che esiste più di un modificatore che le implementa, già integrato in codice nativo. Si introduce inizialmente la tecnica di deformazione di mesh poligonali tramite elasticità discreta, che è stata realizzata, quindi, presenteremo diverse metodologie di deformazione. Illustreremo poi come modellare, creare ed editare delle mesh in Blender. Non ci soffermeremo su dettagli puramente dettati dall'interfaccia utente, cercheremo invece di addentrarci nei concetti e nelle strutture teoriche, allo scopo di avere le basi logiche per definire una Add-on che risulti veramente efficace e utile all'interno del sistema di modellazione. Approfondiremo la struttura di due modificatori chiave per la deformazioni di mesh : Lattice Modifier e Mesh Deform Modifier che implementano una metodologia di tipo space-based. Infine ci concentreremo sulla parte di scripting Python in Blender. Daremo un'idea delle strutture dati, dei metodi e delle funzioni da utilizzare per interagire con l'ambiente circostante, con i singoli oggetti ed in particolare con le Mesh e daremo un esempio di script Python. Andremo infine a descrivere l'implementazione della deformazione elastica mediante add-on Python in Blender.

II. TECNICHE DI MESH EDITING

La deformazione interattiva di una mesh triangolare è un argomento impegnativo, dal momento che le formulazioni matematiche complesse devono essere nascoste da un'interfaccia utente intuitiva e devono essere implementate in maniera sufficientemente efficace e robusta per permettere una perfetta interattività. Introduciamo il problema della deformazione di mesh partendo dal caso continuo. La deformazione di una data superficie $\mathcal{M}$ nella superficie desiderata $\mathcal{M}'$ è descritta matematicamente attraverso una funzione di spostamento d che associa ad ogni punto $p \in \mathcal{M}$ un vettore di spostamento $d(p)$. Attraverso questo vettore si mappa la superficie data $\mathcal{M}$ nella sua versione deformata $\mathcal{M}'$.

$$\mathcal{M}' := \{\, p + d(p) \mid p \in \mathcal{M}\}. \tag{1}$$

Sia M la mesh che rappresenta la discretizzazione della supeficie $\mathcal{M}$, allora, per questa mesh triangolare discreta M, la funzione di spostamento d è completamente definita attraverso i vettori di spostamento $d_i = d(v_i)$ relativi ai vertici della mesh originale $v_i \in M$. L'utente può controllare la deformazione impostando lo spostamento di $\overline{d_i}$ su un insieme di punti $v_i \in \mathcal{H} \subset \mathcal{M}$ chiamati *handle* e considerando un altro insieme vincolato $\mathcal{F} \subset \mathcal{M}$ tenuto fisso durante la deformazione (lo chiameremo *fixed*) :

$$d(v_i) = \overline{d_i},\ \forall v_i \in \mathcal{H}, \tag{2}$$

$$d(v_i) = 0\,,\ \forall v_i \in \mathcal{F}\,. \tag{3}$$

In Figura 1 possiamo osservare come si comportano le diverse zone rispetto alla loro definizione e al vettore di spostamento.

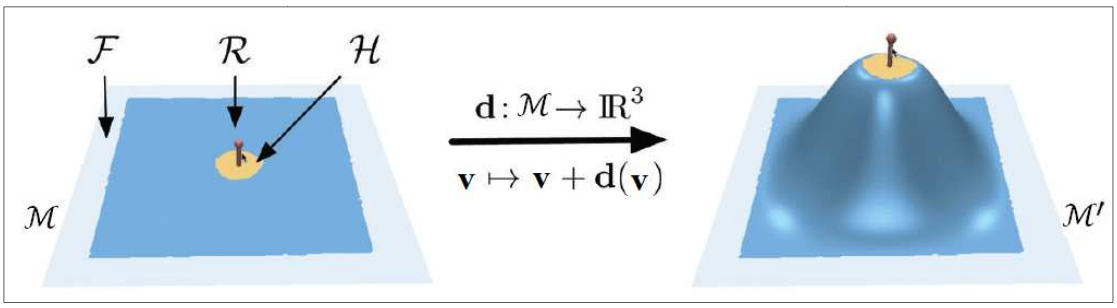

FIGURA 1 REGIONI DI PUNTI E DEFORMAZIONE (1)

Il problema ora è come determinare i vettori di spostamento d_i per tutti i vertici non vincolati $v_i \in \mathcal{R} = \mathcal{M}/(\mathcal{H} \cup \mathcal{F})$ tale che la mesh deformata risultante rispecchi le aspettative dell'utente.

Dopo una presentazione dell'ambito teorico su cui si è basato il lavoro di tesi si presenterà una panoramica sulle differenti tecniche di deformazione prendendo a riferimento lo scrupoloso overview sulle tecniche di deformazione di mesh poligonali in (1).

II.1 Deformazione elastica discreta

I metodi di deformazione surface-based sono un'alternativa sia ai metodi di rigging che a quelli di caging. Ricordiamo che i metodi di tipo rigging prevedono l'utilizzo di uno scheletro che, associato ad una mesh, permette di controllarla e animarla; i metodi di caging prevedono invece l'utilizzo di griglie in cui immergere la mesh per poterla deformare (un esempio è l'operatore Lattice in Blender già ampliamente trattato). L'idea di base di un approccio variazionale consiste nel misurare la qualità di una superficie basandosi su un tipo di energia basata sulla curvatura. Nell'Add-on è stato introdotto sia l'approccio classico basato su Laplacino che un nuovo tipo di formulazione con approccio variazionale introdotto nell'articolo **deformation by discrete elastica** (2). Questo approccio migliora la classica deformazione a superficie Laplaciana utilizzando la curvatura totale che fornisce una deformazione qualitativamente migliore per i corpi elastici. In generale un'energia di curvatura può essere espressa utilizzando le curvature principali di una superficie: curvatura Media ($H = k_1 + k_2$), curvatura Gaussiana ($K = k_1\ k_2$) e curvatura Totale ($k_1^2 + k_2^2$). Il rapporto tra la curvatura e l'energia di bending appare per la prima volta, in una formulazione esplicita, nell'Elasticità di Eulero dove le curve minimizzano l'integrale di curvatura quadratica. Sia $\mathcal{M}$ una superficie definita da due dimensioni, parametrizzata da una funzione

$$X: \Omega \epsilon \mathbb{R}^2 \ \rightarrow \mathcal{M} \in \mathbb{R}^3 \ , dove\ \Omega\ \text{è}\ un\ dominio\ di\ riferimento\ aperto.$$

Le superfici sono governate da un'energia di bending sulla superficie della forma:

$$E(\mathcal{M}) \ = \ \int_{\mathcal{M}} \alpha + \ \beta(H - H_0)^2 - \gamma K \quad d\mathcal{M} \ , \tag{4}$$

Dove H_0 denota la curvatura spontanea, posto $\alpha = H_0 = 0$ e $\beta = 1$, $\gamma = 2$ questo modello si riconduce a quello della Energia di curvatura totale

$$E_T(\mathcal{M}) = \int_{\mathcal{M}} (H^2 - 2K)\, d\mathcal{M} = \int_{\mathcal{M}} ({k_1}^2 + {k_2}^2)\, d\mathcal{M}, \qquad (5)$$

Che approssima l'energia di bending di una superficie piana multidimensionale. In questa seconda formulazione (5) le curvature principali k_1 e k_1 dipendono in maniera non lineare dalla superficie $\mathcal{M}$. Chiamiamo le superfici minimizzate (4) col nome di superfici elastiche perché generalizzano le famose curve elastiche di Eulero. Questa energia risulta invariante su una superficie per movimenti rigidi e cambio uniforme di fattore di scala. Inoltre in (5) il primo temine rappresenta l'energia di Wilmore (3) e il secondo termine si può esprimere come un integrale curvilineo della curvatura geodetica sul contorno $\partial\mathcal{M}$ (4), ciò implica che la curvatura Gaussiana di $\mathcal{M}$ dipende solo da un collare di vicini di $\partial\mathcal{M}$. In più resta costante su superficie con contorno fisso e normali sul contorno fisse ed è una delle ragioni prinipali perché di solito si sceglie di utilizzare l'energia di Wilmore. Però quando una deformazione agisce su qualche regione interna di $\mathcal{M}$, quest'ultima avrà i contorni fissi ma cambieranno le normali, in più una deformazione di solito cambia la topologia della superficie, cambiando quindi anche la sua curvatura Gaussiana. Per queste ragioni il secondo termine (5) non può essere trascurato nel processo di ottimizzazione dell'energia e va incluso nel processo di deformazione. Inoltre sia l'energia totale $E_T(\mathcal{M})$ che l'energia di bending $E_B(\mathcal{M})$ sono invarianti sotto tutte le trasformazioni Möbious cosa che deve essere garantita anche nel caso discreto. L'energia Totale permette di definire la forza elastica interna dell'oggetto espressa come δE, una derivata prima varazionale dell'energia potenziale di deformazione.

II.1.1.1 Energie di deformazione

La deformazione di un corpo non-rigido è una modifica della forma o della dimensione di un oggetto conseguente all'applicazione di forze esterne. Una deformazione è definita elsastica se la forma non deformata (o di riferimento) si ristabilisce completamente una volta rimosse tutte le forze esterne. Nel caso contrario è definita inelastica (non elastica).

Non è stata implementata alcuna deformazione inelastica, utilizzeremo invece le deformazioni elastiche di un modello non rigido durante la deformazione, trascurando poi la fase responsabile di recuperare la forma di riferimento. Sia $\mathcal{M}$ la superficie di riferimento, parametrizzata dalla funzione $X: \Omega \epsilon \mathbb{R}^2 \rightarrow \mathcal{M} \epsilon \mathbb{R}^3$. Una deformazione è una funzione d che mappa $\mathcal{M}$ su un modello deformato $\mathcal{M}'$, aggiungendo per ogni punto $X(u,v) \in \mathcal{M}$ un vettore di spostamento $d(u,v)$ tale che $\mathcal{M}' = X'(\Omega)$, $\mathrm{X}' = \mathrm{X} + \mathrm{d}$.
Un'approssimazione ragionevole per un'energia elastica su strutture a guscio sottile, che misura l'allungamento(stretching) e la piegatura (bending), è:

$$\int_\Omega k_s \, \|I' - I\|^2 + k_b \|II' - II\|^2 \, dudv, \tag{6}$$

dove I (I') e II (II') rappresentano la prima e la seconda forma fondamentale per $\mathcal{M}$($\mathcal{M}'$), k_s e k_b pesano le norme delle matrici e determinano la resistenza all'allungamento e alla piegatura. Per dettagli vedere (1). Le norme di matrice in questa funzione la rendono decisamente non lineare quindi di solito si linearizza questa funzione obiettivo sostituendo la prima e la seconda forma fondamentale con le derivate parziali del primo e del secondo ordine della funzione di spostamento d (5). La Thin Plate Energy o di piegatura è data da :

$$E_B(d) = \frac{1}{2} \int_\Omega k_b \, (\|d_{uu}\|^2 + 2\|d_{uv}\|^2 + \|d_{vv}\|^2) dudv. \tag{7}$$

L'energia di allungamento o Membrane Energy è definita da

$$E_M(d) = \frac{1}{2} \int_\Omega k_s \, (\|d_u\|^2 + \|d_v\|^2) dudv. \tag{8}$$

La deformazione di una superficie non rigida ha bisogno di entrambe queste energie. Per trovare la parametrizzazione della superficie $\mathcal{M}$, Ω è scelta per essere uguale alla superficie originale $\mathcal{M}$, in modo tale che $d = \mathcal{M} \rightarrow \mathbb{R}^3$ sia definito a partire dall'insieme $\mathcal{M}$ stesso. Come conseguenza lo stesso operatore Laplaciano Δ rispetto alla parametrizzazione X si trasforma nell'operatore di Laplace-Beltrami $\Delta_{\mathcal{M}}$ rispetto all'insieme di riferimento $\mathcal{M}$. quindi quando la parametrizzazione è isometrica vale:

$$E_B(d) \cong \frac{1}{2} \int_{\mathcal{M}} K_b \, \|\Delta_{\mathcal{M}} d\|^2 \, d\mathcal{M} \, , \tag{9}$$

e

$$E_M(d) \cong \frac{1}{2} \int_{\mathcal{M}} K_s \, \|\nabla_{\mathcal{M}} d\|^2 \, d\mathcal{M} \, . \tag{10}$$

In cui $E_M(d)$ è soggetta ai vincoli di contorno. La minimizzazione di queste funzioni, che agisce efficacemente applicando il calcolo variazionale, è definita dalla loro equazioni di Eulero-Lagrange che sono:

$$\Delta_{\mathcal{M}}{}^2 \, d = 0, \tag{11}$$

da (10), e

$$-\Delta_{\mathcal{M}} d = 0 \, , \tag{12}$$

da (8). Ora chiamiamo E_C la deformazione variazionale, proposta in (5), formata dalla combinazione tra allungamento e piegatura data da

$$-\mathrm{k_s} \, \Delta_{\mathcal{M}} d + \mathrm{k_b} \, \Delta_{\mathcal{M}}{}^2 \, d = 0 \, . \tag{13}$$

Ma la linearizzazione appena descritta in (10) e (11) , può essere molto costosa rispetto alla dimensione degli artefatti creati, utilizzeremo dunque una minimizzare dell'energia curvatura totale $E_T(\mathcal{M})$ in $E_T(\mathcal{M}) = \int_{\mathcal{M}} (H^2 - 2K) \, d\mathcal{M} = \int_{\mathcal{M}} ({k_1}^2 + {k_2}^2) \, d\mathcal{M}$, (5) in che porta all'equazione di Eulero –Lagrange:

$$\Delta_{\mathcal{M}} H(d) + 2H(d)\big(H^2(d) - K(d)\big) = 0 \, , \tag{14}$$

soggetta alle naturali condizioni di contorno, come proposto in(2). Per essere ben posta richiede due condizioni indipendenti al contorno. Per condizioni naturali al contorno intendiamo che nessuna condizione di continuità è implicata dalla parte esterna incidente al contorno di $\mathcal{M}$. Notiamo che $\sqrt{H^2 - K}$ è una differenza parziale della curvatura principale, quindi anche nel caso discreto deve essere garantito che $H^2 - K \geq 0$.

II.2 ALTRE TECNICHE DI DEFORMAZIONE

Discuteremo di due classi di deformazioni di forma : deformazioni surface-based e deformazioni space-based. Tutti i metodi che descriveremo sono tecniche di deformazione che richiedono la soluzione di un sistema di equazioni lineari solo, in generale, per minimizzare una certa energia di deformazione quadratica. Questi metodi sono avvantaggiati dal fatto che i sistemi lineari possono essere risolti molto efficientemente. Tuttavia possono portare a risultati non intuitivi per deformazioni su larga scala. Le tecniche di deformazione non lineari permettono di superare queste limitazioni, riducendo al minimo, in maniera più accurata, le energie non lineari, che però richiedono schemi numerici più complessi.

II.2.1 SURFACE-BASED DEFORMATION

In una deformazione di tipo suface-based la funzione di spostamento $d: \mathcal{M} \rightarrow \mathbb{R}^3$ è formulata sulla superficie originale $\mathcal{M}$ e si trova a partire dalla mesh triangolare M. I metodi basati su superficie offrono un elevato grado di controllo poiché ogni vertice potrebbe essere vincolato individualmente. D'altro canto la robustezza e l'efficienza dei calcoli sono legate fortemente alla complessità della maglia e alla qualità della superficie triangolare originale.

II.2.1.1 PROPAGAZIONE DELLA TRASFORMAZIONE

Un approccio semplice per le deformazione è quello che lavora propagando la trasformazione definita dall'utente all'interno della regione handle (Figura 1). Dopo aver specificato la regione di supporto R è possibile applicare ad una qualsiasi regione handle $\mathcal{H}$ in R una trasformazione lasciando che se ne occupi l'utente utilizzando un'adeguata interfaccia grafica. La sua trasformazione $T(x)$ sarà poi propagata e smorzata all'interno della regione di supporto, portando ad una fusione armoniosa tra la regione handle trasformata $\mathcal{H}' = T(\mathcal{H})$ e la regione fissata $\mathcal{F}$. Questa fusione è controllata da un parametro scalare $s: \mathcal{M} \rightarrow [0,1]$ che vale 1 nell'handle $\mathcal{H}$ (deformazione completa) e 0 nella regione fixed $\mathcal{F}$ e si muove armoniosamente tra 1 e 0 nella regione di supporto. Un

modo di costruire questo parametro scalare, proposto in (1), è trovare le distanze $dist_F(p)$ e $dist_H(p)$ da v alle regioni $\mathcal{F}$ e $\mathcal{H}$ rispettivamente, e definire

$$s(p) = \frac{dist_{\mathcal{F}}(p)}{dist_{\mathcal{F}}(p)+dist_{\mathcal{H}}(p)}. \quad (15)$$

Le distanze possono essere ad esempio geodetiche della superficie (6) ma anche euclidee dello spazio(7). Un altro modo per trovare il parametro potrebbe ad esempio essere quello di calcolarlo come un valore armonico sulla superficie, ad esempio $\Delta s = 0$. Il problema di questo metodo è che la propagazione della trasformazione in base alla distanza non da il risultato geometricamente più intuitivo.

II.2.1.2 Deformation Shell-Based

Minimizzando le energie di deformazione basate su vincoli fisici possono essere modellate delle deformazioni di superficie d più intuitive con vincoli geometrici prescritti $d(p_i) = \overline{d_i}$. Assumiamo che la superficie si comporti come una pelle o un foglio che viene tirato o piegato con la stessa energia delle forze che agiscono su di esso. Matematicamente, questo comportamento può essere catturato da un'energia funzionale che penalizza sia lo stretching sia il bending. Assumiamo per le seguenti derivazioni che $\mathcal{M}$ e $\mathcal{M}'$ siano superfici parametriche lisce , cioè , attraverso ricavate dalle funzioni :

$$p: \Omega \to \mathbb{R}^3 \text{ e } p': \Omega \to \mathbb{R}^3 . \quad (16)$$

Allo stesso modo la funzione di spostamento è definita da $v: \Omega \to \mathbb{R}^3$. Siano $I(u,v) e\ II(u,v)$ la prima e la seconda forma fondamentale. Quando la superficie è deformata da $\mathcal{M}$ a $\mathcal{M}$' le forme fondamentali cambiano da I a I' e da II a II', la differenza delle forme fondamentali può essere usata come un'energia sulla mesh (a guscio sottile) in grado di misurare stretch e bending (8)(2) . I parametri ks e kb sono utilizzati per controllare la resistenza all'estensione e alla piegatura, rispettivamente. In uno scenario di modellazione si deve minimizzare l'energia elastica $\int_\Omega k_s \|I' - I\|^2 + k_b \|II' - II\|^2 \, dudv$, (6) $-\mathrm{k_s}\ \Delta_{\mathcal{M}} d + \mathrm{k_b}\ \Delta_{\mathcal{M}}{}^2\, d = 0$. (13)soggetta ai vincoli di

deformazione imposti dall'utente. Come mostrato in Figura 1 di solito questo significa fissare una certa regione $\mathcal{F}$ e definire lo spostamento per una regione handle $\mathcal{H}$.
Tuttavia la minimizzazione dell'energia non lineare $E(\mathcal{M}')$ ha costi computazionali troppo alti per applicazioni interattive, pertanto si semplifica sostituendo alla differenza delle forme fondamentali le derivate parziali della funzione di spostamento d(2)(9). Questo porta alla seguente energia thin-shell:

$$E(d) = \iint_{\Omega} k_s(\|d_u(u,v)\|^2 + \|d_v(u,v)\|^2) + k_b(\|d_{uu}(u,v)\|^2 + \|d_{vu}(u,v)\|^2 + \|d_{vv}(u,v)\|^2)dudv \quad (17)$$

In cui usiamo la notazione $d_u = \frac{\partial d}{\partial u}$ e $d_{uv} = \frac{\partial^2 d}{\partial u \partial v}$ per denotare le derivate parziali. Per minimizzare (17) efficacemente applichiamo un calcolo variazionale che produrrà un'equazione corrispondente di Eulero-Lagrange (13).
Pertanto per minimizzare (17) dobbiamo risolvere l'equazione differenziale a derivate parziali appena enunciata. A questo punto possiamo tornare dalla superficie parametrica continua ad una mesh discreta con facce triangolari e sostituire la formula continua della PDE con la analoga discreta. Il Laplaciano potrebbe essere discretizzato nel seguente modo:

$$\Delta d_i \coloneqq \frac{1}{2A_i}\sum_{v_j \epsilon N_1(v_1)}(cot_{\alpha_{ij}} - cot_{\beta_{ij}})(\mathrm{d_j} - \mathrm{d_i}) \quad (18)$$

Dove $d_i = d(p_i)$ rappresenta lo spostamento del punto p_i, e dunque, se esiste, del corrispondente vertice v_i. Mentre il bilaplaciano è dato da

$$\Delta^2 d_i = \Delta(d_i)\,. \quad (19)$$

Quindi possiamo discretizzare rispetto alla tipologia di vertice:

$$\begin{aligned} -k_s\Delta d + k_b\Delta^2 d &= 0 \quad p_i \epsilon \mathcal{R}\,, \\ d_i &= \bar{d}_i \quad p_i \epsilon \mathcal{H}, \\ d_i &= 0 \quad p_i \epsilon \mathcal{F}. \end{aligned} \quad (20)$$

Queste condizioni possono essere formulate come un sistema lineare di equazioni , la cui incognite sono gli spostamenti $d_1 .. d_n$ dei vertici free $\mathcal{R}$:

$$[-k_s L + -k_b \mathrm{L}^2]\begin{pmatrix} d_1^T \\ \vdots \\ d_n^T \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}. \qquad (21)$$

Nel capitolo IV, relativo all'implementazione utilizzeremo questa strategia di deformazione, daremo uno sguardo in dettaglio sulla rappresentazione parametrica (IV.1) e sull'utilizzo del sistema (21) (IV.2), indicando anche una possibile ottimizzazione (IV.3).

II.2.1.3 Multi-Scale Deformation

La deformazione shell-based della sezione precedente fornisce una deformazione basata sulla fisica. La performance interattiva si ottiene semplificando l'energia shell-based non lineare $E(\mathcal{M}')$ in modo tale che, per ottenere la superficie $\mathcal{M}'$, debba essere risolto solo un sistema lineare. Tuttavia, come conseguenza di questa linearizzazione, il metodo non gestisce correttamente i dettagli superficiali. La rotazione locale dei dettagli geometrici è un comportamento intrinsecamente non lineare e pertanto non può essere modellato da un tecnica puramente lineare. Un modo per conservare meglio i dettagli geometrici, utilizzando comunque un approccio di deformazione lineare, è quello di utilizzare le tecniche multi-scale. L'idea principale delle deformazioni multi-scale è quello di scomporre l'oggetto in due bande di frequenza utilizzando le tecniche di smoothing e fairing. Le basse frequenze corrispondono alla forma globale, mentre le alte frequenze corrispondono ai dettagli. L'obiettivo che ci si pone è quello di deformare le frequenze basse (forma globale), preservando poi i dettagli ad alta frequenza , con conseguente deformazione multi- scale desiderata. Come prima cosa viene calcolata una rappresentazione a bassa frequenza della data superficie $\mathcal{M}$, rimuovendo le alte frequenze ed ottenendo così una superficie liscia di base $\mathcal{B}$. I dettagli geometrici saranno quindi $D = \mathcal{M} \ominus \mathcal{B}$. Questo permette di ricostruire la superficie originale $\mathcal{M}$ aggiungendo i dettagli geometrici sulla superfice di base $\mathcal{M} = \mathcal{B} \oplus D$. Gli operatori speciali $\ominus$ e $\oplus$ sono

chiamati rispettivamente operatore di decomposizione e operatore di ricostruzione del frame work multi-scale (1).

II.2.1.4 Coordinate differenziali

Mentre la deformazione multi-scala è in effetti uno strumento per migliorare le deformazioni di forma attraverso il mantenimento dei dettagli, la generazione di una tale gerarchia può diventare molto pesante per modelli geometricamente e topologicamente complessi. Per evitare la decomposizione esplicita di tipo multi-scala un'altra classe di metodi modifica le proprietà della superficie differenziale invece di modificare le coordinate spaziali, ricostruendo una superficie deformata avente le coordinate differenziali desiderate. Descriviamo prima due delle rappresentazioni più usate, Gradiente e Laplaciano e come ricavare la superficie deformata manipolando le coordinate differenziali.

II.2.1.5 Deformazioni Gradient-based

I metodi di deformazione basati su gradiente eseguono la deformazione manipolando i gradienti della superficie originale per poi trovare la superficie deformata che corrisponde al gradiente del campo di destinazione rispetto ai minimi quadrati. Per manipolare i gradienti, dobbiamo prima di tutto considerare una funzione lineare a tratti $f: \mathcal{M} \to \mathbb{R}$ sulla mesh originale M che è definita dai suoi valori f_i ai vertici della mesh. Il suo gradiente $\nabla f: \mathcal{M} \to \mathbb{R}^3$ è un vettore costante a tratti, cioè un vettore costante $g_T \epsilon \mathbb{R}^3$ per ogni triangolo T. Se invece di una funzione scalare f viene considerata la funzione a coordinate lineari a tratti $p: \mathcal{M} \to \mathbb{R}^3$, $v_i \to p_i$, allora il gradiente di una faccia T è una matrice costante Jacobiana 3x3:

$$\nabla \mathrm{v}|_T = \begin{bmatrix} \nabla p_x|_T \\ \nabla \mathrm{p}_y|_T \\ \nabla p_z|_T \end{bmatrix} =: J_T \epsilon \mathbb{R}^{3x3}. \qquad (22)$$

Le righe di J_T sono i gradienti delle coordinate x,y e z per la funzione v dentro un triangolo T.

I gradienti J_T della faccia sono poi modificati moltiplicandoli per una matrice M_T di dimensioni 3x3 che rappresenta la rotazione, la scalatura o il taglio desiderati fornendo il rispettivo gradiente :

$$J'_T = M_T J_T. \tag{23}$$

Lo step che ci resta da affrontare è trovare la nuova posizione dei vertici v'_i e dunque del corrispondente punto p'_i, in modo tale che i gradienti $\nabla \mathrm{p}'|_T$ della mesh deformata siano il più simile possibile ai gradienti desiderati J'_T. Nel continuo il problema analogo potrebbe essere quello di cercare la funzione $f: \Omega \longrightarrow \mathbb{R}$ che corrisponde meglio ad un dato valore di gradiente g. Che corrisponde a risolvere l'equazione di Eulero-Lagrange:

$$\Delta f = div \mathrm{g}\,, \tag{24}$$

che deve essere risolta per la funzione f ottimale (1). Sostituendo f con le coordinate x,y,z dei vertici v_i appartenenti alla deformazione e discretizzando l'equazione di Eulero-Lagrange $\Delta f = div \mathrm{g}$, (24) , troviamo il sistema:

$$L \begin{pmatrix} {p'_1}^T \\ \dots \\ {p'_n}^T \end{pmatrix} = \begin{pmatrix} div J'(v_1) \\ \dots \\ div J'(v_n) \end{pmatrix} \tag{25}$$

Il sistema va risolto singolarmente per ogni coordinata, nel termine noto si ha la divergenza dei gradienti modificati su x, y e z.

II.2.1.6 Deformazioni laplacian-based

La seconda classe di modi di deformazione basate su coordinate differenziali sono quelle basate su Laplaciano. La metodologia è molto simile all'editing su gradiente appena esposto, con la sola differenza che qui andiamo ad agire sul laplaciano, quindi lavoriamo su vertici e non su facce. Prima devono essere calcolate le coordinate iniziali $\delta_i = \Delta(\mathrm{p_i})$, le manipoliamo e infine troviamo le nuove coordinate p'_i, che corrispondono alle coordinate Laplaciane desiderate. L'impostazione continua del problema consiste nel minimizzare

$$E(p') = \iint_\Omega \|\Delta \mathrm{p}'(\mathrm{u},\mathrm{v}) - \delta'(u,v)\|^2 du dv\,, \tag{26}$$

Che porta all'equazione di Eulero-Lagrange

$$\Delta^2 p' = \Delta \delta'. \tag{27}$$

Su una mesh triangolare M discreta risolvere tale equazione si riconduce alla soluzione di un sistema bi-laplaciano che deve essere risolto per le coordinate x, y, z dei vertici deformati v'_i :

$$L^2 \begin{pmatrix} v'^T_1 \\ \dots \\ v'^T_n \end{pmatrix} = \begin{pmatrix} \Delta\delta'^T_1 \\ \dots \\ \Delta\delta'^T_n \end{pmatrix} \tag{28}$$

Anche in questo caso devono essere imposte le condizioni al contorno per $\mathcal{F}$ *e* $\mathcal{H}$. Tra le deformazioni shell-based e Laplacian-based c'è un collegamento interessante . Cerchiamo di lasciare per un momento le trasformazioni locali $\delta_i \longmapsto \delta_i'$ e calcoliamo invece le nuove coordinate p_i'dai Laplaciani originali δ_i, ad esempio risolvendo $\Delta^2 p' = \Delta\delta$ imponendo nuovamente i vincoli $p'_i = \overline{p_i}$ per $p_i \in \mathcal{H} \cup \mathcal{F}$. Utilizzando le due identità $p' = p + d$ e $\delta = \Delta \mathrm{p}$ si scopre che quest'ultima equazione alle derivate parziali equivale all'equazione di Eulero–Lagrange $\Delta^2 d = 0$ dell'approccio shell-based. Di conseguenza, i due metodi sono equivalenti fino a quando modellano le rotazioni locali di dettagli geometrici o di coordinate differenziali, rispettivamente, impiegando sia una tecnica multi-scale sia trasformazioni locali di Laplaciani. Un'altra conseguenza è che questa modifica Laplaciana produce una deformazione C^1 continua, rispetto alla deformazione basata sui gradienti che invece è C^0continua.

II.2.2 SPACE-BASED DEFORMATIONS

L'approccio space-based non lavora direttamente sulla superficie come accade nelle deformazioni suface based ma immerge la superficie in uno spazio che la contiene completamente e deforma quest'ultimo. Il legame che si impone tra la superficie e lo spazio che la circonda implica che una deformazione su quest'ultimo si ripercuota sulla mesh. Quindi, rispetto alle deformazioni surface-based, che definiscono il vettore di spostamento sulla superficie $\mathcal{M}$, le deformazioni space-based definiscono una funzione di spostamento d: $\mathbb{R}^3 \to \mathbb{R}^3$ che deforma l'intero spazio incorporato in $\mathbb{R}^3$ e con si deforma implicitamente anche la superficie $\mathcal{M}$. La deformazione non richiede calcoli sulla maglia triangolare M e quindi tali metodi sono meno compromessi dalla complessità e dalla qualità di $\mathcal{M}$ rispetto ai metodi surface-based. Questi problemi vengono evitati usando questo tipo di deformazione poichè si deforma lo spazio circostante ed implicitamente

l'oggetto al suo interno. Questo approccio assume una funzione di deformazione trivariata $d: \mathbb{R}^3 \to \mathbb{R}^3$ per trasformare tutti i punti della superficie originale $\mathcal{M}$. Fino a che la funzione di deformazione spaziale d non dipende da una particolare rappresentazione della superficie, può essere utilizzata per deformare tutti i tipi di rappresentazioni superficiali esplicite.

II.2.2.1 Lattice-Based FreeForm Deformation

La deformazione classica free form deformation rappresenta la deformazione spaziale attraverso una funzione trivariata spline (10):

$$d(u,v,w) = \sum_i \sum_j \sum_k \delta c_{ijk} N_i(u) N_j(v) N_k(w)\ , \tag{29}$$

dove N_i sono funzioni base B-spline e $\delta c_{ijk} = (c'_{ijk} - c_{ijk})$ sono gli spostamenti dei punti di controllo c_{ijk}. Rendendo più semplice la formulazione (1) avremo

$$\delta c_l \coloneqq \delta c_{ijk} \tag{30}$$

e

$$N_l(u) = N_l(u,v,w) \coloneqq N_i(u) N_j(v) N_k(w). \tag{31}$$

Questo ci permette di riscrivere l'equazione come

$$d(u) = \sum_{l=1}^{n} \delta c_l N_l(u)\ . \tag{32}$$

Ogni vertice originale $v_i \epsilon M$ ha un corrispondente valore di parametro $u_i = (u_i, v_i, w_i)$ tale che $v_i = \sum_l c_l N_l(u)$. Il vertice viene poi trasformato da $v'_i = v_i + d(u_i)$ che può essere calcolato in maniera efficiente quando $N_l(u_i)$ rimane costante. La deformazione può essere controllata manipolando le posizioni dei punti di controllo, vale a dire, prescrivendo gli spostamenti dei punti di controllo δc_l. Questo, tuttavia, può diventare noioso per griglie di controllo più complesse. Inoltre, il sostegno della deformazione può essere difficile da prevedere perché è determinato come intersezione di un supporto di funzioni a base volumetrica con la superficie $\mathcal{M}$. Un'interfaccia basata su manipolazione diretta,

permettendo all'utente di specificare spostamenti dei punti della superficie ancorchè di punti di controllo c_l, semplifica il processo di deformazione. Dato un insieme di vincoli di spostamento $d(u_i) = \overline{d_i}$ per $\{p_1, \dots, p_m\} = \mathcal{H} \cup \mathcal{F}$ va risolto un sistema lineare per i movimenti δc_l riguardanti i control point:

$$\begin{bmatrix} N_1(u_1) \dots N_n(u_1) \\ \dots \\ N_1(u_m) \dots N_n(u_m) \end{bmatrix} \begin{pmatrix} \delta c_1 \\ \dots \\ \delta c_n \end{pmatrix} = \begin{pmatrix} \overline{d_1} \\ \dots \\ \overline{d_m} \end{pmatrix}. \tag{33}$$

Questo sistema di dimensioni (m x n) potrebbe essere sia sottodeterminato che sovradeterminato. Questo produce una soluzione ai minimi quadrati (least squares) e minima norma (least norm) che minimizza l'errore nei vincoli $\sum_i \left\| d(u_i) - \overline{d_i} \right\|^2$ così come la quantità di movimento del control point $\sum_l \|\delta c_l\|^2$. Mentre da un lato si riesce a produrre una soluzione ben definita, dall'altro presenta due inconvenienti : in primo luogo, in un'impostazione sovradeterminata dei vincoli di spostamento non può essere soddisfatta esattamente, ma solo arrivando ad una soluzione dettata dai minimi quadrati. In secondo luogo, nel settaggio sottodeterminato, i restanti gradi di libertà sono determinati minimizzando i movimenti del control point, invece di utilizzare un'ottimizzazione per renderlo il più liscio possibile, come accadeva nell'utilizzo di deformazioni surface-based. Si riporta alla sezione III.1.2.2 in cui andremo ad osservare il funzionamento del Lattice Modifier in Blender, basato sulla tecnica appena descritta.

II.2.2.2 Caged based free form deformation

Le tecniche cage-based possono essere considerate una generalizzazione di quelle basate su lattice. Invece di controllare la deformazione con un reticolo regolare viene utilizzata una cosiddetta *control cage* (gabbia di controllo). Di solito la gabbia è una mesh triangolare grossolana che racchiude l'oggetto che deve essere modificato; questo permette alla gabbia di adattarsi meglio alla forma e alla struttura dell'oggetto incorporato rispetto a quello che faceva il reticolo regolare nella deformazione Lattice. I vertici p_i della mesh originale $\mathcal{M}$ possono essere rappresentati come combinazione lineare dei vertici di controllo della gabbia c_1 attraverso

$$p_i = \sum_{l=1}^{n} c_l \varphi_l(p_i), \tag{34}$$

dove i pesi $\varphi_l(p_i)$ sono coordinate baricentriche generalizzate e le funzioni φ_l corrispondono alle funzioni spline N_l nella corrispettiva definizione per la formulazione basata su Lattice. Una volta che i pesi $\varphi_l(p_i)$ sono stati computati, l'oggetto può essere deformato andando a muovere i vertici della gabbia $c_l \longrightarrow c_l + \delta c_l$ e computando il movimento per vertice come

$$d(p_i) = \sum_{l=1}^{n} \delta c_l \varphi_l(p_i), \tag{35}$$

che lo definisce rispetto agli spostamenti dei vertici di controllo δc_l. Tali spostamenti sono conseguenti ad una deformazione che lavora allo stesso modo descritto per la deformazione lattice-based, sostituendo $\varphi_l(p_i)$ a $N_l(u_i)$. Anche se questo metodo risulta molto più comodo in termini di griglia di controllo, questa soluzione non garantisce una deformazione migliore della precedente. Nella sezione III.1.2.3 descriveremo l'utilizzo di questo approccio alla deformazione per la definizione del Mesh Deform modifier in Blender, nella sezione vengono descritti ampiamente i pro e i contro di questa soluzione.

II.2.2.3 Radial Basis Functions

Nel caso di deformazioni basate sulle superfici, si realizzano risultati di alta qualità interpolando i vincoli di spostamento dell'utente con una funzione di deformazione d che minimizza alcune energie. Su questa base si può ricavare una funzione di deformazione spaziale trivariata con interpolazione d: $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ che minimizza tali energie.
Ad un livello più astratto, il problema è quello di trovare una funzione che interpola alcuni vettori di spostamento $\overline{d_i}$ conosciuti per la posizione p_i. A questo scopo si può utilizzare una funzione a base radiale trivariata (1), poiché le funzioni a base radiale (RBFs) sono note per essere molto adatte per questo tipo di problema (11).

III. LA SUITE BLENDER

Blender è una suite open source di tools per la computer grafica. La suite è stata scelta sia per via della sua estrema configurabilità sia per il suo supporto multipiattaforma. Fino alla versione 2.25 la programmazione in Python sull'applicazione era estremamente limitata, oggi è ormai quasi totalmente integrata. Prova ne è il fatto che, ad esempio, nella versione 2.67, utilizzata per la creazione dell'add-on, tutto quello che è interfaccia utente è stato sviluppato in Python, arrivando addirittura al punto che, se ci si posiziona con il mouse su un qualsiasi pulsante dell'interfaccia, comparirà il nome del file Python che ne implementa le funzionalità. La possibilità di testare gli script direttamente sull'applicazione e di poter interagire con quasi tutto l'ambiente circostante rende infine il tool di sviluppo Python uno strumento molto potente. In questo capitolo daremo un overview su come il tool gestisce le mesh e in che modo è possibile interagire con esse lato user-interface . Nella seconda parte del capitolo dettaglieremo quali sono le strutture principali rese disponibili dalla suite sia per definire una user interface completa e funzionale, sia per interagire con le mesh poligonali.

III.1 INTERAGIRE CON LE MESH IN BLENDER

Ogni mesh poligonale è costruita a partire da tre strutture basilari: vertici, spigoli e facce, la loro rappresentazione strutturale è in coordinate cartesiane e la loro semplicità crea, come sappiamo, le fondamenta per tutti i modelli. A partire da questi tre elementi dunque è possibile cambiare la geometria della nostra mesh. Proprio per questa ragione la suite rende disponibile una modalità apposita di lavoro: la modalità Edit. In questa modalità è possibile interagire su ognuno di questi elementi. Proprio in questa modalità è possibile ad esempio associare i vertici in gruppi. Questa funzionalità è stata fondamentale per la realizzazione dell'add-on che, come introdotto nella Sezione II.1, dovrà lasciare all'utente la possibilità di definire un insieme di vertici handle H ed un insieme di vertici fixed F sulla mesh poligonale di riferimento M. Partendo da questa scelta applicherà la deformazione sull'insieme $R = M/(H \cup F)$ che è il corrispettivo discreto dell'insieme $\mathcal{R} \in \mathcal{M}$ descritto nell'introduzione della Sezione II.1. In contrapposizione all'Edit Mode è possibile lavorare

sull'oggetto intero in Object Mode. Tutti i modificatori (modifier) esistenti in Blender lavorano di default in questa modalità, lasciando inalterata la geometria dell'oggetto, cioè l'insieme dei suoi vertici, dei suoi edge e delle sue facce. Fatto saldo la possibilità di applicare in maniera definitiva i cambiamenti andando, in questo caso, a modificare la geometria dell'oggetto. Di seguito andremo a vedere come interagire sulla mesh sia in modalità Edit che tramite l'uso di modifier.

III.1.1 MANIPOLAZIONE DI MESH IN EDIT MODE

In Blender tutti gli oggetti condividono qualche proprietà, tutti hanno un loro tipo e ogni tipo ha le sue specificità. Le mesh in particolare sono, per loro natura, uno degli oggetti più complessi che si possano trovare in Blender. Molte delle modifiche apportabili alla forma delle mesh sono applicabili solo in Edit Mode. Questa modalità infatti viene attivata in automatico subito dopo la creazione di una mesh. Quando lavoriamo in Edit Mode vediamo la mesh nella sua forma base, quindi vedremo i vertici, gli spigoli e le facce che la compongono. In più è sempre visibile il centro dell'oggetto. Non si può lavorare contemporaneamente su tutti e tre gli elementi della geometria, anche se la morfologia della mesh segue i cambiamenti che apportiamo ad ognuno di essi. L'elemento base di una mesh è il vertice, e possiamo definirlo come un punto o una posizione nello spazio 3D.

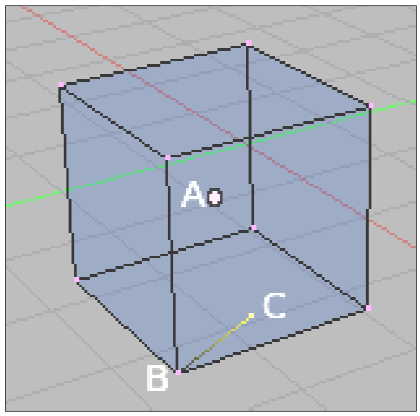

FIGURA 2 - VERTICI DI UN CUBO (12)

In Figura 2 possiamo osservare il punto centrale etichettato come "A" ed i vertici "B" e "C". Inoltre vediamo come sia possibile definire un nuovo vertici senza che ci sia bisogno di collegarlo ad una faccia. Uno spigolo per sua natura connette sempre due vertici con una

linea retta, mentre, la faccia è la struttura a più alto livello in una mesh e con il suo impiego si può costruire la superficie di qualsiasi oggetto. Le facce sono le uniche che vengono poi visualizzate in fase di rendering. Una faccia è definita come l'area compresa fra tre o quattro vertici, comprendente uno spigolo per ogni lato. L'uso di triangoli è sempre preferibile dato che sono sempre piatti e facili da calcolare. Bisogna invece prestare molta attenzione nel maneggiare facce a quattro spigoli, poiché internamente sono divise in due triangoli. Questo tipo di facce sono funzionali solo se risultano particolarmente piatte (tutti i punti giacciono su un piano immaginario), e convesse (nessuno degli angoli ai vertici è maggiore o uguale a 180 gradi). Ricadono in questo caso, ad esempio, le facce di un cubo. Va comunque ricordato che un'area compresa fra tre o quattro vertici, anche se evidenziata da spigoli, può non costituire una faccia. Un'area di questo genere risulterà semplicemente trasparente e inesistente nell'immagine che sarà frutto del rendering.

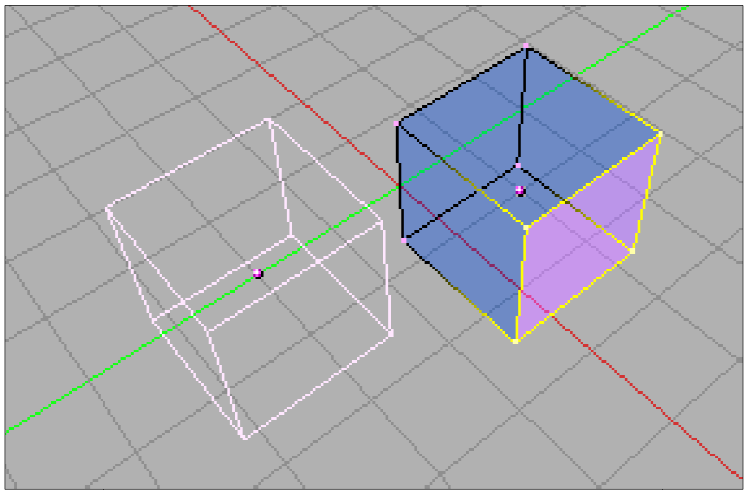

FIGURA 3 - PRIORITÀ DI SELEZIONE IN EDIT MODE (12)

Uno dei punti focali per l'editing di mesh è la finestra View 3D in Edit Mode che consta dei seguenti componenti :

- Mesh Tool, menu sul lato sinistro della finestra
- Trasform, menu sul lato destro della finestra
- Mesh, menu della barra in basso della finestra – raccoglie molti degli strumenti di editing

- Select, menu della barra in basso della finestra – raccoglie quasi tutti i tool di selezione
- Altri menu contestuali, ognuno dei quali è fortemente specializzato.

III.1.1.1 LOOPS SU SPIGOLI E FACCE

I loop sono insiemi di Spigoli o di Facce che formano anelli continui. Come mostrato in figura 2, la riga superiore (1 - 4) mostra una vista “solid”, la riga inferiore (5 – 8) una vista “wireframe” degli stessi loops.

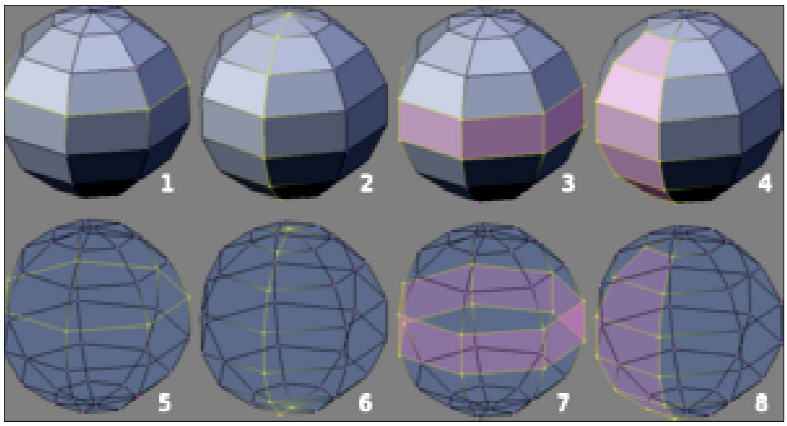

FIGURA 4 - EDGE E FACE LOOPS (12)

Si noti poi che i loops 2 e 4 non percorrono l'intero modello, ma si fermano ai cosiddetti poli. Questo accade perché non è possibile continuare un loop dopo un polo. I poli sono definiti come vertici che sono connessi a tre, cinque o più spigoli. Per conseguenza logica i vertici connessi ad esattamente uno, due o quattro spigoli non sono poli.
I loops che non terminano in poli sono ciclici (1 e 3), partono e terminano nello stesso vertice e dividono il modello in due partizioni. I loops possono rappresentare un modo veloce e potente di lavorare con regioni continue e specifiche di una mesh e sono un prerequisito per l'animazione di personaggi di tipo vivente (organic character).

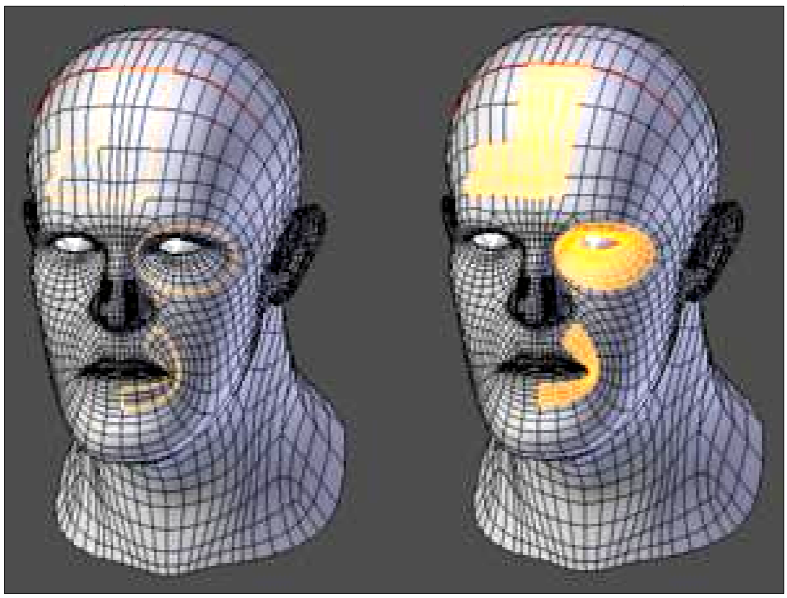

FIGURA 5 - LOOPS NELLA MODELLAZIONE ORGANICA (12)

I Loop su Spigoli sono un concetto importante specialmente nella modellazione organica e nell'animazione dei personaggi. Quando sono usati correttamente, permettono di costruire delle zone che risulteranno particolarmente naturali se sottoposte ad un algoritmo di suddivisione. Inoltre si deformano molto bene nell'animazione. Prendiamo l'esempio in figura 3: I loop su spigoli seguono i contorni naturali e le linee di deformazione della pelle e i muscoli sottostanti sono più densi nelle zone che subiranno una maggiore deformazione se il personaggio si muove, per esempio a livello delle spalle o delle ginocchia. I Face loops sono una logica estensione degli Edge Loops e comprendono le facce tra due Edge Loops, come mostrato nei loops 3 e 4 in figura 2.

III.1.1.2 ELEMENTI NASCOSTI

Tra le funzionalità che Blender fornisce spicca la possibilità di nascondere alcune parti di mesh o oggetti interi, fatto saldo che è possibile renderli visibili appena si renda necessario. Questo è un enorme vantaggio per un utilizzo maneggevole e pulito delle nostre view, soprattutto quando si sta lavorando su un modello complesso con centinaia di vertici.

III.1.1.3 UNDO & REDO

Blender ha un sistema globale di undo&redo, che permette di cancellare le ultime operazioni compiute (di selezione o di editazione). Questo strumento lavora in Object mode, quindi permetterà di annullare il complesso della sessione compiuta in Edit Mode con un unico passo. Viene mantenuto, comunque, uno storico di utilizzo anche per l'Edit Mode. Lo storico di editazione è mantenuto durante il passaggio tra le modalità Edit e Object, per lo stesso tempo per cui l'oggetto è attivo. Questo significa che si può passare dalla modalità Object a quella di Edit senza perderlo se non abbiamo cambiato l'oggetto attivo (selezionato). Un'operazione di "Undo" su una mesh potrebbe essere molto impegnativa a livello di memoria. Per chi sta utilizzando una macchina con RAM limitata, Blender fornisce la possibilità di settare il numero massimo di passi di undo da salvare. Il range ammissibile è tra 1 e 64 , di default è 32.

III.1.1.4 VERTEX GROUPS

Una mesh potrebbe essere composta da un'enorme quantità di vertici connessi. Lavorando ad un progetto si potrebbe aver bisogno di riutilizzare parti della mesh o di nascondere delle ampie aree per poter lavorare sui dettagli. Nel caso di un modello particellare, potremmo aver bisogno di definire zone che evolvono in maniera differente. Infine, nel nostro caso, abbiamo bisogno di definire delle zone che saranno dei punti di riferimento per un processo di deformazione. Per necessità di questo tipo Blender ci permette di raggruppare i vertici utilizzando i vertex groups. Nel caso di creazione di bones per l'animazione o per definire le pose di un personaggio, Blender dà la possibilità di creare automaticamente i vertex group implicati nel movimento. Essi dunque identificano dei sotto-componenti di una mesh. Per definire una regione di vertici come vertex group non si deve far altro che selezionare i vertici implicati e associarli al gruppo tramite interfaccia grafica, una volta fatto è possibile lavorarci su in maniera isolata senza aver bisogno di creare un oggetto separato. L'uso di vertex groups rende anche semplice eliminare o duplicare più volte una parte di una mesh. Prendiamo come esempio la modellazione di un blocchetto Lego™. Il blocchetto più semplice consiste di una base e di un punto (o "nipple"). Per creare un blocco a quattro punti quindi, definiremo un vertex group che

racchiude i vertici del “nipple” e, in Edit mode, lo duplicheremo andando a formare il nostro blocchetto. Un altro uso dei vertex groups è la definizione di uno scheletro (armature). Se si cerca di animare la propria mesh e farla muovere tramite algoritmi di ridding, dovremmo definire uno scheletro che è formato da un raggruppamento di ossa (bones). Quando un osso viene spostato deforma tutti i vertici a cui è associato. Quindi, quando muoveremo l’osso relativo al braccio, esso muoverà i vertici relativi al braccio e non quelli relativi alle gambe. In questo modo parti della mesh possono essere allungate e mosse mentre altre parti restano fisse. I vertex group possono essere utilizzati anche come elementi particellari , il “weight painting” del gruppo definirà quante particelle usciranno e a che velocità. Ricordiamo che i capelli sono una particella statica, quindi potremmo definire un vertex group chiamato “Scalpo” ed usarlo per dire a Blender di far fuoriuscire i capelli dallo scalpo. Un altro grande uso dei Vertex Groups è il riutilizzo delle selezioni effettuate.

III.1.1.4.1 CREARE UN VERTEX GROUP

Un oggetto appena creato non ha nessun vertex group associato, dunque non è definita alcuna zona che lega i suoi vertici. Non tutti gli oggetti possono definire dei vertex group, la motivazione è banalmente il fatto che non posseggono vertici. I vertex groups sono visibili solo quando un oggetto è in Edit Mode e proprio in questa modalità si può interagire con il menu apposito. E’ buona norma essere sicuri che i vertici siano stati veramente assegnati ad un gruppo utilizzando i pulsanti Select e Deselect presenti sotto la lista dei vertex groups.

III.1.1.4.2 ESEMPIO D’USO DEI VERTEX GROUP

Consideriamo un armadietto da cucina che consiste in tre piani verticali (due lati e un posteriore), una base e un piano di lavoro, un telaio di sportello, uno sportello, un pomello e due cerniere. Si potrebbe impostare il progetto in modo tale da poter, in futuro, essere in grado di modellare l’apertura degli sportelli. Potremmo infatti aver creato un armadietto con una singola porta e subito dopo, aver scoperto di doverlo modificare inserendo un’ulteriore porta. Sarebbe auspicabile poter copiare il design di un unico pomello per

usarlo su ulteriori oggetti in cucina. Ho dunque bisogno di definire almeno tre vertex group : “base”, “porta” e “pomello”. Prendiamo ad esempio la duplicazione di parti : Se cerchiamo di associare a questo armadietto un modello a due porte, volendo utilizzare la duplicazione del pomello precedentemente creato, dobbiamo semplicemente:

1) Selezionare il mobiletto e passare in Edit Mode.
2) Assicurarci che nessun vertice sia selezionato.
3) Selezionare il vertex group di nome “pomello” dal menu apposito.
4) Attivare la selezione.
5) Duplicare la sub-mesh. I vertici saranno copiati, selezionati e agganciati.
6) Utilizzare la mesh appena creata.

E’ importante ricordare che i vertici duplicati appartengono allo stesso gruppo degli originali, quindi bisogna assegnare questo nuovo pomello ad un suo gruppo che chiameremo ad esempio “pomello.L”. A questo proposito, in Blender esiste una convenzione di nomi che ci permette di compiere azioni relative ad un gruppo che è la controparte destra o sinistra di un altro oggetto. Se il nome termina con ”.L” o con ”.left” e la sua controparte con ”.R” o ”.right”, Blender sarà in grado di replicare semplicemente le modifiche apportate ad uno solo dei due gruppi.

III.1.1.4.3 PESO (WEIGHT)

Di default, ogni vertice presente in un vertex group ha un peso di 1.00. Se un vertice appartiene a più gruppi, ha un peso coordinato. Quando è influenzato da un osso o da altri oggetti, si muove di una quantità proporzionale al suo peso; più pesante è il vertice, meno si muoverà. Quindi un vertice inserito in due gruppi, in ognuno dei quali ha peso 1, si muoverà della metà rispetto ad un vertice appartenente ad un unico gruppo con il medesimo peso. Questo sistema di pesi fornisce una deformazione realistica di una mesh modificata tramite bones, per esempio, attorno all’area della spalla, dove alcuni dei vertici appartengono sia al gruppo che identifica il torace, sia a quello che identifica il braccio. Possiamo settare il peso di tutti i vertici in un gruppo usando il controllo numerico del peso, Weight Painting che permette di sfumare senza problemi i singoli pesi dei vertici in modo che le mesh si deformino in maniera più adeguata.

III.1.2 MANIPOLAZIONE DI MESH CON MODIFIERS

I Modificatori sono operazioni che vengono applicate ad un oggetto in maniera non distruttiva. Con i modifier si possono automatizzare molti effettti che manualmente potrebbero essere tediosi da realizzare (come definire una surface subdivision), il tutto senza andare ad influenzare la topologia di base dell'oggetto. I modificatori lavorano cambiando la visualizzazione ed il rendering dell'oggetto, ma non la sua geometria, che resterà comunque visibile in Edit Mode. E' possibile aggiungere diversi modificatori ad un singolo oggetto formando un vero e proprio stack di modifier. Se si vogliono rendere permanenti sulla geometria dell'oggetto i cambiamenti imposti dal modificatore si può eseguire l'operazione " Apply".

III.1.2.1 CATEGORIE

In Blender ci sono quattro categorie di modificatori. Di queste, la categoria Generate definisce gli strumenti di creazione che cambiano la struttura dell'oggetto o aggiungono una nuova struttura all'oggetto esistente. Questa categoria contiene:

- Array : Crea un array dalla nostra mesh base e ne ripete la forma
- Bevel : Permette di smussare i bordi della mesh a cui viene applicato, permettendo di controllare quanto e dove la smussatura è applicata alla mesh.
- Boolean : Combina / Sottrae / Interseca la nostra mesh con un'altra.
- Build : Assembla la nostra mesh passo passo mentre facciamo animazione.
- Decimate : Riduce il numero dei poligoni della nostra mesh.
- Edge Split : Aggiunge tagli agli spigoli nella nostra mesh.
- Mask : Permette di nascondere alcune parti della nostra mesh.
- Mirror : Specchia un oggetto rispetto ad uno dei suoi assi, in modo che la mesh risultante sia simmetrica.
- Multiresolution : Scolpisce la nostra mesh per diversi livelli di risoluzione.
- Remesh : può correggere pesantemente la triangolazione nelle mesh o altri problemi che hanno bisogno di un'attenta regolazione.

- Screw : Genera una geometria seguendo un pattern ad elica partendo da un profilo semplice. In maniera simile allo strumento Screw presente nel contesto del mesh editing.
- Skin: Topologia generata automaticamente.
- Solidify : Dà profondità alle facce della mesh.
- Subdivision surface : suddivide la nostra mesh utilizzando un algoritmo semplice o quello di Catmull-Clark.
- Triangulate : Converte tutte le facce in triagoli.

Il gruppo Deform comprende tutti quei modificatori che si occupano di deformazioni di forma su un oggetto e sono disponibili per mesh, often text, curve, superfici e/o griglie lattice. Qui troviamo:

- Armature : Usa le bones per deformare e animare il nostro oggetto.
- Cast : trasforma la forma in una sfera, cilindro o cuboide.
- Cube : Piega il nostro oggetto usando una curva come guida.
- Displace : Deforma il nostro oggetto usando una texture.
- Hook : Aggiunge un collegamento ad un nostro vertice (o control point) per manipolarlo dall'esterno.
- Laplacian Smooth : Permette di ridurre il rumore sulla superficie di una mesh con cambiamenti minimi sulla sua forma.
- Lattice : Usa una griglia Lattice per deformare il nostro oggetto.
- Mesh Deform : Ci permette di deformare il nostro oggetto modificando la forma di un'altra mesh usando una "Mesh Deform Cage" (simile all'uso di un oggetto lattice).
- Shrinkwrap : permette di ridurre /avvolgere il nostro oggetto in/attorno alla superficie di un oggetto mesh.
- Simple Deform : Applica qualche deformazione avanzata al nostro oggetto.
- Smooth : Smussa la geometria di una mesh. Simile allo strumetno smooth nel contesto mesh editing.
- Wrap : Restringe una mesh specificando due punti tra cui la mesh si estende.
- Wave : Deforma il nostro oggetto per formare onde (animate).

- Simulate : il gruppo di modificatori Simulate attivano simulazioni. In molti casi, questi modificatori sono automaticamente aggiunti allo stack dei modificatori ogni volta che un sistema particellare o un simulatore fisico è attivo ed il loro unico ruolo è di definire il posto nello stack dei modificatori usato come dato di base per lo strumento che rappresentano. Generalmente gli attributi di questi modificatori sono accessibili in pannelli separati.
- Cloth: Simula le proprietà di un pezzo di stoffa. E' inserito nello stack dei modificatori quando noi definiamo una mesh come "cloth".
- Collision : simula una collisione tra oggetti.
- Explode : fa saltare la nostra mesh utilizzando un sistema particellare.
- Fluid : L'oggetto è una parte della simulazione di fluido. Modificatore aggiunto quando si designa una mesh come fluido.
- Particle Instance : Crea un'azione di oggetto simile ad una particella ma usando la forma della mesh.
- Particle System: Rappreseta un sistema particellare nello stack, quindi è inserito quando viene aggiunto un sistema particellare all'oggetto.
- Smoke : Simula un fumo realistico.
- Soft Bod : L'oggetto è leggero, elasticizzato. Modificatore aggiunto quando designamo una mesh come un oggetto soft.
- Dynamic Paint : Crea un oggetto o un sistema particellare disegnando un materiale su un altro oggetto.
- Ocean : crea velocemente un oceano realistico ed animato.

Infine, nel gruppo Modify si trovano modifier simili ai Deform Modifiers. La differenza è che non agiscono sul'intera struttura dell'oggetto, ma su sue sottocomponenti, come ad esempio i vertex groups. Di questa categoria fanno parte:

- Mesh Cache : Applica dati di animazione ad una mesh (da file esterno).
- UV_Project : Progetta coordinate UV sulla nostra mesh.
- UV Warp : Cambia dinamicamente le coordinate UV sulla nostra mesh.
- Vertex Weight : Cambia un vertex group della nostra mesh in diversi modi.

III.1.2.2 LATTICE MODIFIER

Questo modifier è l'implementazione dei concetti esposti nella sezione II.2.2.1, dedicata alle deformazioni freeform basate su lattice. Definisce dunque una deformazione space-based. Deforma l'oggetto basandosi sulla forma di una griglia di tipo lattice. Può essere applicato ad un oggetto mesh intero o ad un suo vertex group. Un oggetto Lattice consiste in una griglia non renderizzabile di vertici e la sua maggiore funzionalità è dare una capacità di deformazione extra all'oggetto sotto il suo controllo (attraverso un modifier o lavorando da parent). Gli oggetti soggetti alla modifica possono essere mesh, curve, superfici, testi, lattici e anche sistemi particellari. Il suo punto di forza è la semplicità di utilizzo. Come già descritto nella teoria, per deformazioni di questo tipo, il lato opposto della medaglia è che la deformazione non risulta fisicamente credibile per oggetti con forme non particolarmente cuboidali. Si può usare lo stesso lattice per deformare diverse mesh.

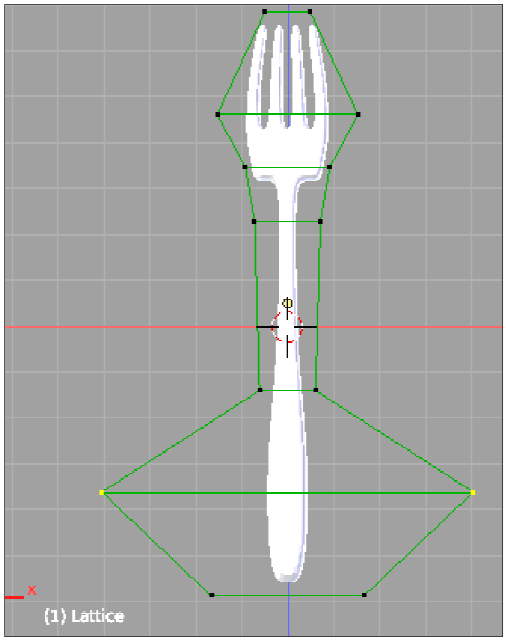

FIGURA 6 - LATTICE MODIFIER (13)

Leggeri cambiamenti all'oggetto mesh, utilizzando questo modifiatore sono resi in maniera immediata e semplice, lasciando oltretutto la struttura della mesh inalterata.

III.1.2.3 MESH DEFORM MODIFIER

Questo modificatore implementa una deformazione di tipo cage-based che, come già descritto in teoria generalizza quella basata su lattice (II.2.2.2). Difatti il modificatore Mesh Deform permette di usare una qualsiasi mesh chiusa come una gabbia deformante attorno ad un'altra mesh. La gabbia può essere di qualsiasi forma e non solo cuboidale come invece accade per il modificatore Lattice. Il modificatore Mesh Deform è abbastanza semplice da usare ma può essere molto lento a calcolare la mappatura corretta della gabbia deformante in rapporto all'oggetto deformato. La propietà Object rappresenta il nome della mesh da usare come gabbia deformante. Anche in questo caso c'è la possibilità di utilizzare i vertex group per limitare l'influenza della cage. L'operazione con cui l'oggetto verrà legato alla mesh che fungerà da cage è il Bind. Questa operazione permette al modificatore di assumere la mesh scelta come gabbia all'oggetto da deformare, cosicché i vertici della cage fungono da control point per la modifica dell'oggetto connesso. Un problema di questo approccio è che a seconda del settaggio del modificatore, dalla complessità della gabbia o dell'oggetto da modificare, può volerci molto tempo prima che l'operazione di modifica sia completata. Può succedere anche che Blender non risponda alle azioni dell'utente prima che abbia completato il calcolo ed è anche possibile che resti a corto di memoria ed il progamma vada in crash. E' comunque possibile fare l'unbind dissociando il nostro oggetto dall'eventuale gabbia associata. Quando l'unbind è attuato l'oggetto che funge da gabbia mantiene la sua forma attuale, non ritorna quindi alla sua forma originale. Se si vuole mantenere la forma originale, si deve salvare una copia dell'oggetto prima di alterarlo. Ad ogni modo l'oggetto deformato torna alla sua forma originale, com'era prima di essere legato alla gabbia. Un ulteriore parametro numerico che è possibile applicare al mesh modifier è la precisione, che ci permette di controllare l'accuratezza con la quale la gabbia altera l'oggetto deformato quando i punti sulla gabbia vengono spostati. I valori relativi a questo parametro devono essere compresi tra 2 e 10 ed il valore di default è 5. Aumentare questo valore può incrementare di molto il tempo che il modificatore necessita per calcolare lo spostamento, ma permetterà di avere una mappatura della deformazione molto più accurata. Il valore di precisione non può essere modificato una volta attivato il Bind. Infine un'ulteriore proprietà attivabile per questo modificatore è

il dynamic, se lo utilizziamo stiamo indicando al modificatore che deve tenere conto di deformazioni e cambiamenti che non sono il risultato diretto del modificatore Mesh Deform. Con Dynamic attivo, tutti gli altri componenti che modificano la mesh (come per esempio altri modificatori o Shape keys) sono tenuti in considerazione quando viene fatto il bind su una gabbia, aumentando la qualità della deformazione. Di default è disattivata per preservare l'uso di memoria e il tempo di calcolo. Anche questa opzione non è modificabile una volta che il bind è stato effettuato. Le modifiche apportate alla deform mesh cage o gabbia deformante si rifletteranno solo nell'oggetto deformato quando la gabbia è in modalità Edit, quando invece è in modalità Object la gabbia può essere scalata e distorta, ma non avrà alcun effetto sull'oggetto deformato. Quando la gabbia viene legata ad un oggetto da deformare, questa deve circondare tutte le parti dell'oggetto che devono essere influenzate dalla gabbia. Una volta che la gabbia è stata "bindata" può essere spostata lontano dall'oggetto deformato in object mode. Quando poi si utilizzerà in Edit Mode e se ne modificherà la forma, essa altererà l'oggetto deformato pur non circondandolo.

III.1.2.3.1 DISTANZA TRA LA CAGE E LA SUPERFICIE DELL'OGGETTO

La distanza tra la cage e l'oggetto da deformare influisce sulla quantità di cambiamento impartita all'oggetto deformato quando viene applita una modifica (in Edit mode). Quando la gabbia è più lontana dall'oggetto deformato, la quantità di cambiamento impartita è sempre inferiore e meno locale rispetto ad una specifica area dell'oggetto deformato. Viceversa, quanto la gabbia è più vicina all'oggetto deformato, tanto maggiore sarà la sua influenza, anche localmente su una specifica area sull'oggetto deformato.

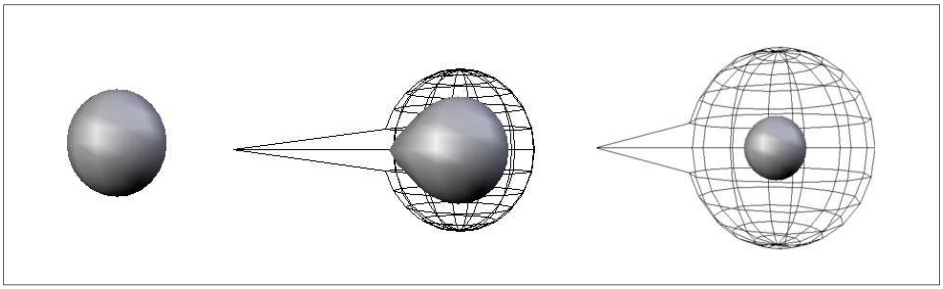

FIGURA 7 - DEFORMAZIONE RISPETTO ALLA DISTANZA CAGE-OBJECT (14)

Quelli riportati in figura 7 sono esempi dell'effetto di diverse distanze all'interno della cage tra l'oggetto da deformare e il bordo della stessa. La prima immagine mostra una normale sfera UV non deformata. L'immagine centrale mostra la stessa sfera UV circondata da una cage a lei molto vicina, di conseguenza una modifica alla cage causerà una grande deformazione. L'ultima immagine mostra la deformazione di un oggetto quando la cage è più lontana. Si può notare che l'alterazione dell'oggetto è decisamente più moderata, tenuto conto che il vertice è stato spostato nella medesima direzione e quantità.

III.1.2.3.2 SUPPORT MULTIRES

Dalla versione 2.49, Blender è in grado di gestire sia il modificatore Mesh Deform che la proprietà Multires ovvero la gestione della risoluzine multipla - con la limitazione che il modifier Mesh Deform funziona solo quando il livello MultiRes era attivo al momento del bind. Questo significa che quando si sta lavorando ad un livello di MultiRes diverso da quello che si stava utilizzando durante il binding, il modificatore non avrà alcun effetto sull'oggetto deformato. Naturalmente, è possibile aggiungere più modificatori Mesh Deform relativi ai diversi livelli di risoluzione.

III.1.2.4 IMPLEMENTAZIONE DEL MESH DEFORM MODIFIER

Il metodo di implementazione del modificatore Mesh Deform (da Blender versione 2.46) segue le linee guida tracciate da (15) con l'introduzione delle "coordinate armoniche". L'uso di queste coordinate ha introdotto molti vantaggi nel controllare le deformazioni. L'idea nasce dall'associare l'oggetto da deformare in una determinata posizione rispetto ad una mesh triangolare chiusa che fungerà da gabbia. L'oggetto viene quindi "legato" alla gabbia calcolando un peso $g_i(p)$ per ogni vertice della gabbia. C_i ,cioè la posizione dei control point appartenenti alla gabbia, sono valutati rispetto alla posizione di ogni punto p dell'oggetto. Quando i vertici della gabbia vengono spostati in nuove posizioni C'_i i punti deformati sono computati da

$$p' = \sum_i g_i(p) C'_i \text{ i} \qquad (36)$$

Un esempio è mostrato in Figura 8 (b). Le funzioni peso $g_i(p)$ sono note come coordinate a valor medio. Le coordinate di valore medio derivano da coordinate baricentriche generalizzate e hanno ottime proprietà utili all'uso in deformazioni cage-based. Tuttavia, possiamo dire che questo tipo di coordinate hanno un inconveniente se utilizzate per le articolazioni di personaggi, come illustrato dal personaggio mostrato in Figura 8. Notiamo come i vertici della gabbia modificati per la gamba sulla sinistra nella Figura 8 (e) influenzano significativamente la posizione dei vertici dell'oggetto nella gamba destra.

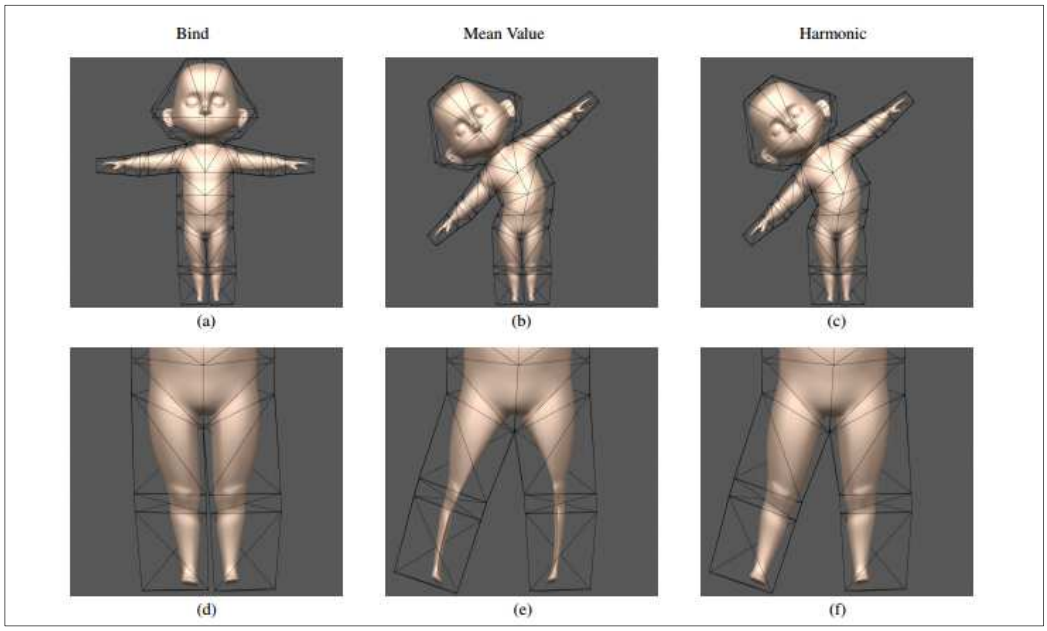

FIGURA 8 - EFFETTO DI COORDINATE A VALOR MEDIO ED ARMONICHE(15)

Ciò si verifica perché le coordinate a valor medio si basano su distanze euclidee (costanti) tra i vertici della gabbia e i punti dell'oggetto. Poiché la distanza tra i vertici della gabbia modificata e i punti dell'oggetto nelle gambe sulla destra è relativamente piccola nella posa fissata, l'influenza è relativamente grande. Notiamo anche che lo spostamento dei punti di questo oggetto è in direzione opposta allo spostamento dei vertici gabbia. Questo si verifica poiché le coordinate a valore medio sono negative, tale comportamento è inaccettabile per l'articolazione di personaggi in produzione cinematografica. Il comportamento indesiderato sopra illustrato si verifica perché le coordinate a valor medio mancano di due proprietà che sono essenziali per definire la perfetta articolazione su un personaggio, vale a dire :

- **Località Interna:** Informalmente, le coordinate dovrebbero ricadere tra l'oggetto originale e la gabbia, quindi dovrebbero essere calcolate come una funzione della distanza tra i vertici della gabbia e i punti dell' oggetto, dove la distanza è misurata dentro la gabbia.
- **Non negatività:** Come illustrato nelle figure 8(e) e 8 (b), se un punto appartiene ad un oggetto le cui coordinate relative ai vertici della gabbia risultano negative, il punto dell'oggetto e il vertice della gabbia si muoveranno nel senso opposto. Per impedire questo comportamento poco intuitivo, sono state introdotte delle coordinate per cui è garantito un valore non negativo per l'interno della gabbia, anche in situazioni fortemente concave.

Nel lavoro realizzato da Pixar[8] si dimostra che le coordinate che possiedono queste due proprietà critiche possono essere prodotte come soluzioni dell'equazione di Laplace. Poiché le soluzioni dell'equazione di Laplace sono genericamente definite come funzioni armoniche, questo tipo di coordinate sono chiamate coordinate armoniche e la deformazione che generano è chiamata deformazione armonica (harmonic deformation).

A differenza delle coordinate a valor medio, le coordinate armoniche non possiedono una espressione in forma chiusa e devono essere approssimate utilizzando un solutore numerico. Ci sono due potenziali svantaggi nell'utilizzare un solutore numerico: il tempo e la precisione. Nel caso delle articolazioni di un personaggio quando l'animatore muove i vertici della gabbia si aspetta che l'oggetto venga deformato in tempo reale. Per tali deformazioni completamente interattive le coordinate devono essere calcolate in pre-processing, prima di qualsiasi interazione con l'utente. Questa precomputatione è necessaria, anche se esiste una forma di tipo chiuso. Dato che le coordinate devono essere precalcolate, che esista o meno una forma chiusa, il tempo necessario per eseguire il risolutore di coordinate armoniche è accettabile. Il problema fondamentale per quanto riguarda la precisione invece è se sia possibile riprodurre le funzioni lineari utilizzando tempi ragionevoli. Le coordinate armoniche in questo campo hanno dato dei buoni risultati.

III.1.2.5 COORDINATE ARMONICHE

Formalizziamo la discussione del paragrafo precedente definendo tutte le proprietà necessarie per l'utilizzo nell'elaborazione finale ottima per quanto riguarda l'articolazione

di un personaggio. Iniziamo col considerare la costruzione delle coordinate di valor medio che vengono ricavate partendo da un interpolatore a valor medio. Per calcolare un valore interpolante per ogni punto interno p si consideri ogni punto x sul perimetro, si moltiplichi f (x) per la distanza reciproca tra x e p, quindi la media su tutti gli x (vedi figura 9 A). Questa definizione mette in chiaro che le coordinate di valor medio coinvolgono distanze in linea retta a prescindere dalla visibilità di x da p. Nelle articolazioni di un personaggio la gabbia spesso ha grandi concavità e un interpolante più funzionale dovrebbe rispettare la visibilità di un vertice nella gabbia da un punto sull'oggetto. Per costruire tale interpolante, possiamo calcolare la media non su tutti i percorsi in linea retta, ma piuttosto su tutti i percorsi browniani a partire da p, dove il valore assegnato a ciascun percorso è il valore di f preso nel punto di tale percorso che si trova al limite del perimetro della gabbia (vedi figura 9 B).

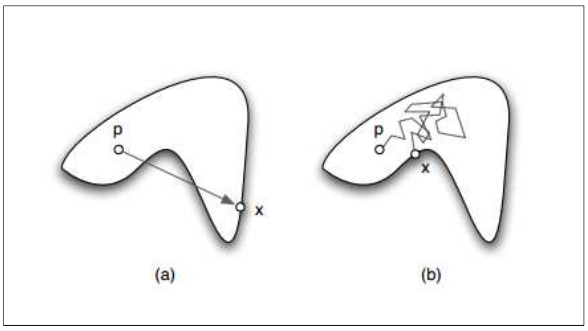

FIGURA 9 - INTERPOLANTE VALOR MEDIO(A) , ARMONICO (B)(15)

In un primo momento, questo interpolante sembra intrattabile per la computazione. Tuttavia l'interpolante così prodotto (in qualsiasi dimensione) in realtà soddisfa l'Equazione di Laplace alle condizioni al contorno date da f. Pertanto, si possono ottenere la coordinate baricentriche generalizzate da una soluzione numerica dell'equazione di Laplace nella gabbia interna. Più formalmente, sia presa la gabbia C come un poliedro in d dimensioni - cioè un volume chiuso (non necessariamente convesso) con un perimetro lineare chiuso a tratti. In due dimensioni, una gabbia è una regione del piano delimitata da un poligono chiuso ed in tre dimensioni una gabbia è una regione chiusa dello spazio delimitato da facce planari (anche se non necessariamente triangolari) . Per ciascuno dei vertici C_i della gabbia si cerca una funzione $h_i(p)$ definita su C e soggetta alle seguenti condizioni :

- Interpolazione: $\mathrm{h_i}(C_i) = \delta i_{,\mathrm{j}}$.
- Uniformità : Le funzioni $\mathrm{h_i}(p)$ sono almeno C^1 risolvibili all'interno della gabbia.
- Non negatività: $\mathrm{h_i}(p) \geq 0$, per tutti i $p \epsilon C$.
- Località Interna : Questo concetto è stato definito imponendo che si ha località interna se, oltre alla non negatività, le funzioni delle coordinate non comprendono punti esterni.
- Riproduzione lineare: Data una funzione arbitraria $\mathrm{f}(p)$, le funzioni delle coordinate possono essere utilizzate per definire un interpolatore $\mathrm{H_{[F]}}(p)$ tale che :

$$\mathrm{H_{[F]}}(p) = \sum_i h_i(p) f(C_i) \tag{37}$$

abbiamo bisogno di $\mathrm{H_{[F]}}(p)$ e per l'esattezza delle funzioni lineari tali per cui $\mathrm{f}(p) = p$, ciò significa che :

$$\mathrm{p} = \sum_i h_i(p) C_i \tag{38}$$

- Invarianza Affine:

$$\sum_i h_i(p) = 1 \quad \forall \ p \epsilon C \tag{39}$$

- Generalizzazione rigorosa delle coordinate baricentriche: quando C è un simplesso, $h_i(p)$ è la coordinata baricentrica di p rispetto a Ci.

Le coordinate a valor medio non posseggono due di queste proprietà, la non negatività e la località interna. Si può dimostrare che le funzioni che generano delle coordinate che soddisfano tutte le sette proprietà possono essere ottenute come soluzioni dell'equazione di Laplace

$$\nabla^2 h_i(p) = 0 \, , p \in Int(C) \tag{40}$$

se le condizioni al contorno sono opportunamente scelte.

Per dare un attimo di luce su come vengono determinate dalle condizioni al contorno consideriamo la costruzione delle coordinate armoniche in due dimensioni. Sarà quindi

possibile poi generalizzare la costruzione in d dimensioni. Le condizioni al contorno appropriate per $h_i(p)$ in due dimensioni sono:

Sia data ∂p che denota un punto sul perimetro ∂C di C, quindi

$$h_i(\partial p) = \Theta_i(\partial p) \quad \forall\, \partial \in \partial C \tag{41}$$

dove $\Theta_i(\partial p)$ è la funzione lineare (invariante) a tratti tali che $\Theta_i(\mathrm{C_j}) = \delta_{ij}$.

III.1.2.6 CONTROLLO INTERNO

Oltre la gabbia stessa, al suo interno possiamo avere ulteriori facce che possono formare una "sotto-mesh" meno ampia e chiusa, consentendo un controllo supplementare su alcune zone dell'oggetto deformato. Queste sotto-mesh possono essere collegate alla gabbia principale o intersecarla. Ad esempio, si potrebbe utilizzare una grande gabbia deformante per un viso intero (per ottenere un effetto da "cartone animato"), con due sub-mesh intorno agli occhi , per controllarli meglio.

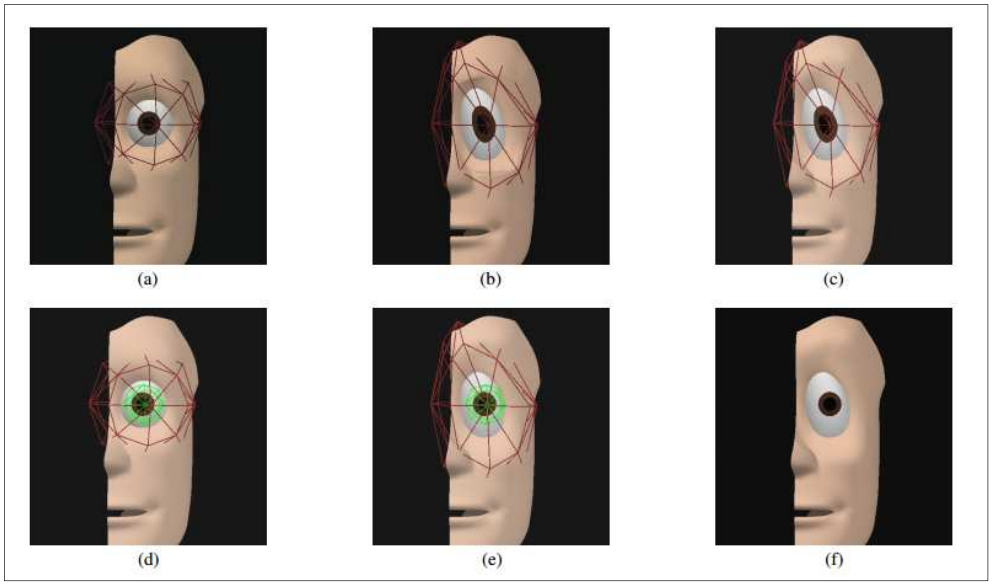

FIGURA 10 - CONTROLLO INTERNO (15)

III.2 IL LINGUAGGIO PYTHON IN BLENDER

Blender dispone di un interprete Python interno ben equipaggiato e funzionale. Questo permette di aggiungere funzionalità all'applicazione scrivendo nuovi script. La versione di Python in Blender 2.67, cioè quella utilizzata per la nostra realizzazione, è la 3.3. Python è un linguaggio di programmazione interpretato, interattivo e orientato agli oggetti. Comprende moduli, eccezioni, ha una gestione dinamica dei tipi, tipi dinamici di alto livello e classi. È stato espressamente progettato per essere usato come linguaggio di estensione per applicazioni che richiedono un'interfaccia programmabile. Per estendere Blender le scelte sono solo due: utilizzare dei plugin binari o utilizzare degli script python. Gli script Python sono più potenti, versatili e più facili da capire oltre che più robusti e sono da preferire rispetto ai primi. D'altro canto, visto che le strutture dati fornite a python per interagire con il mondo Blender sono state costruite ad hoc, il team di sviluppo ha introdotto alcune limitazioni per mantenere la stabilità del sistema. Una forte limitazione è l'impossibilità di poter aggiungere un nuovo modificatore custom da affiancare agli esistenti. Resta comunque la possibilità di creare un surrogato che non verrà affiancato agli esistenti ma che può comunque essere utilizzato. I modificatori infatti sono definiti in codice nativo. La programmazione Python ha avuto delle funzionalità limitate fino a Blender 2.25, poi è stato riorganizzato tutto l'esistente e vi sono stati aggiunti molti moduli nuovi. Questo da un lato ha migliorato la modalità di programmazione introducendo strutture più compatte e funzionali e dall'altro ha reso inusabile tutto il patrimonio di script python che era stato creato . L'evoluzione è comunque continua e ci si deve aspettare una migliore integrazione nelle prossime versioni di Blender. Per l'utilizzo del nostro Add-on abbiamo avuto bisogno di utilizzare delle librerie per interagire con matrici sparse, che per loro natura si adattano favorevolmente a mappare le matrici laplaciane. Per mesh molto grandi infatti, l'uso di matrici di tipo classico, porta a dover istanziare matrici di dimensioni enormi considerando che per ogni vertice dovrebbe essere descritto il legame con uno qualsiasi degli altri vertici della mesh. Dunque sono state importate in Blender le librerie scipy che oltre a definire le matrici sparse forniscono anche i risolutori ottimizzati per lavorare su questi tipi di matrici.

III.2.1 FINESTRA DI TEXT EDITOR

La finestra di Text Editor è perfettamente integrata all'interno del bacino finestre fornito da Blender, infatti è possibile raggiungerla scegliendo la voce TextEditor all'interno del menù dei tipi di Finestra. Una nuova finestra di questo tipo si presenta grigia e vuota con una barra degli strumenti molto semplice. Per renderla editabile basta selezionare Create Text Block dal Menu Text. Tutti gli script aperti durante la sessione sono riportati in una lista, rendendo quindi intuitivo il passaggio da uno script all'altro. Molto meno funzionale invece, devo dire, l'intellisense che è totalmente assente quando vogliamo scrivere direttamente nel TextEditor, chiaramente non manca la possibilità di poter utilizzare un editor esterno specifico per Python. Se abbiamo bisogno di vedere come una certa sequenza di codice si comporterà possiamo comunque usare la Python console. Anche in questo caso la console è totalmente integrata e viene utilizzata come una normale finestra di Blender, quindi la possiamo trovare nell'elenco delle finestre disponibili. La console Python risulta fondamentale soprattutto se si è alle prime armi e si vuole interagire direttamente con le strutture dati e pasticciare un po' col codice. In più la console ci aiuta indicandoci con l'uso di CTRL-space le possibili opzioni per completare le istruzioni. Online inoltre è disponibile il manuale aggiornato all'ultima versione delle librerie. Uno script può dunque essere scritto direttamente nella finestra TextEditor ed eseguito, il risultato comparirà in console. Attenzione però a non confondere le console. La console in cui comparirà l'elaborazione appena avviata e le eventuali stampe a video, è quella da cui è stato avviato il software Blender. Nel caso di Windows, fino a Windows 7, Blender viene lanciato lasciando aperta la console di avvio, nel caso di sistemi unix, se si vuole vedere questa console bisogna far partire il programma da un terminale. Il file scritto è direttamente distribuibile non avendo bisogno di compilazione .

III.2.2 MANIPOLAZIONE DI MESH CON PYTHON

Il modulo bmesh fornisce l'accesso a strutture di dato bmesh di Blender. Questa API permette di accedere alle api interne di blender per l'editing delle mesh, fornendo i dati di connettività della geometria e l'accesso alle operazioni di editing come split, separate, collapse e dissolve. Le caratteristiche esposte sono molto simili alle API C native. Il

modulo bmesh è scritto per essere autonomo, le uniche eccezioni sono per mathutils che viene utilizzato per le posizioni di vertici e normali e la conversione dei dati di mesh da e verso il tipo bpy.types.Mesh.

III.2.2.1 ACCESSO AI DATI MESH

Ci sono 2 modi per accedere ai dati BMesh, è possibile creare un nuovo ogetto BMesh convertendo una maglia da bpy.types.BlendData.meshes o accedendo alla mesh corrente passando dalla modalità Edit, con l'utilizzo delle funzioni :

`bmesh.types.BMesh.from_mesh`

e

`bmesh.from_edit_mesh.`

Quando attuiamo la conversione in modo esplicito dalla mesh, python possiede i dati, vale a dire che la mesh esiste solo mentre python ne detiene un riferimento e lo script ha la responsabilità di rimetterlo in un data-block di tipo Mesh quando le modifiche da apporare sono terminate. Si noti che a differenza di bpy, una BMesh non corrisponde necessariamente ai dati nel file .blend aperto, una BMesh può essere creata, modificata e liberata senza che l'utente abbia mai visto o avuto accesso ad essa. A differenza della modalità di Edit il modulo bmesh può usare più istanze di BMesh alla volta. Bisogna comunque fare attenzione quando si trattano più istanze di BMesh, in quanto i dati contenuti nel reticolo possono utilizzare un sacco di memoria, è dunque buona pratica essere sicuri di svuotare la memoria dedicata ad una mesh ormai inutilizzatta chiamando `bmesh.types.BMesh.free()` che rimuoverà tutti i dati della mesh immediatamente e disabiliterà ulteriore accessi.

III.2.2.2 MANTENERE UNO STATO COERENTE

Quando si modella in Blender ci sono alcune assunzioni da fare circa lo stato della mesh:

- La parte di geometria nascosta (hide) non è selezionata.
- Quando si seleziona uno spigolo, anche i suoi vertici sono selezionati.
- Quando è selezionata una faccia, vengono selezionati i corrispondenti spigoli e vertici.
- Non esistono duplicati di spigoli/facce.

- Le facce hanno almeno 3 vertici.

Per dare agli sviluppatori una buona flessibilità queste convenzioni non sono forzate, comunque gli strumenti custom creati devono lasciare la mesh in uno stato valido per evitare che il resto degli strumenti possano comportarsi in maniera non corretta. Eventuali errori che nascono dal non seguire queste convenzioni sono considerati bug nello script, non bug in Blender.

III.2.2.3 SELEZIONE / FLUSHING

Come detto sopra, è possibile creare uno stato di selezione non valido (per esempio selezionando uno stato e poi de-selezionando uno dei suoi vertici), il modo migliore per risolvere questo problema è fare il flush dopo l'esecuzione di una serie di modifiche. Questa convalida lo stato di selezione.

III.2.2.4 ESEMPIO DI SCRIPT

L'esempio seguente presuppone di avere un oggetto di tipo mesh selezionato. Si prenderà la mesh attiva e se ne preleverà la rappresentazione BMesh . All'interno della geometria della mesh Bmesh si incrementa di 1 il valore della coordinata x. Una volta completata la modifica si riporta la bmesh nell'oggetto mesh.

```
1.     import bpy
2.     import bmesh
3.     me = bpy.context.object.data
4.     bm= bmesh.new()
5.     bm.from_mesh(me)
6.     for v in bm.verts:
7.          v.co.x +=1.0
8.     bm.to_mesh(me)
9.     filename = os.path.join(os.path.dirname(bpy.data.filepath),"addon-
move3.py")
10.    exec(compile(open(filename).read(), filename, 'exec'))
```

III.2.3 ACCESSO AI DATI

Python accede ai dati di Blender nello stesso modo del sistema di animazione e dell'interfaccia utente, il che implica che qualsiasi impostazione che può essere modificata

tramite un pulsante può anche essere modificata da Python. L'accesso ai dati dal file .blend attualmente caricato è fatto attraverso il modulo bpy.data. Questo dà l'accesso ai dati della libreria. Es:

```
>>> Bpy.data.objects
<bpy_collection[3], BlendDataObjects>
>>> Bpy.data.scenes
<bpy_collection[1], BlendDataScenes>
>>> Bpy.data.materials
<bpy_collection[1], BlendDataMaterials>
```

III.2.3.1 Accesso a Collections

Noteremo che sia un indice che una stringa può essere utilizzato per accedere ai membri di una collections. A differenza dei dizionari di Python, sono accettabili entrambi i metodi, ma bisogna fare attenzione al fatto che l'indice di un membro può cambiare durante l'esecuzione di Blender.

```
>>>list (bpy.data.objects)
[bpy.data.objects ["Cube"], bpy.data.objects ["Plane"]]
>>> Bpy.data.objects ['Cube']
bpy.data.objects ["Cube"]
>>> Bpy.data.objects [0]
bpy.data.objects ["Cube"]
```

III.2.3.2 Accesso agli attributi

Una volta che si dispone di un data-block, come ad esempio un materiale, oggetto, gruppi ecc,si può accedere ai suoi attributi allo stesso modo con cui agiremmo utilizzando l'interfaccia grafica. In realtà, il tooltip per ogni pulsante mostra anche l'attributo di Python, che può aiutare a trovare le impostazioni da cambiare in uno script.

```
>>> Bpy.data.objects [0]. Nome
'Camera'
>>> Bpy.data.scenes ["Scene"]
bpy.data.scenes ['scene']
>>> Bpy.data.materials.new ("MyMaterial")
bpy.data.materials ['MyMaterial']
```

III.2.3.3 CREAZIONE / RIMOZIONE DEI DATI

Coloro che hanno familiarità con altre Api Pyton potrebbero restare sorpresi del fatto che i nuovi blocchi dati nella api bpy non possono essere creati chiamando la classe:

```
>>> Bpy.types.Mesh ()
Traceback ():
  File "<blender_console>", line 1, in <module>
TypeError: bpy_struct.__new__ (type): prevede un solo argomento
```

Questa scelta è stata presa intenzionalmente nella progettazione delle API. Difatti le Api Blender / python non possono creare dati di blender che esistono al di fuori dal database principale di Blender (accessibile tramite bpy.data), perché questi dati sono gestiti da Blender (salvatagio / load / undo / append). I dati vengono aggiunti e rimossi tramite metodi sulle collezioni in bpy.data, ad esempio:

```
>>> mesh = bpy.data.meshes.new (name = "MyMesh")
>>> Print (mesh)
<bpy_struct, Mesh("MyMesh.001")>
>>> Bpy.data.meshes.remove (mesh)
```

III.2.3.4 CONTESTO

Mentre è utile essere in grado di accedere ai dati direttamente per nome o come una lista, è molto più comune operare su selezioni dell'utente. Il contesto è sempre disponibile tramite l'istruzione `bpy.context` e può essere utilizzato per ottenere l'oggetto attivo, la scena, il

settaggio del tool altri attributi. Casi di uso comune sono `Bpy.context.object`, `Bpy.context.selected_objects`, `Bpy.context.visible_bones`.
Si noti che il contesto è di sola lettura. Questi valori non possono essere modificati direttamente, anche se possono essere modificati eseguendo funzioni API o utilizzando le API di dato. Quindi la riga 1 dell'esempio seguente genererà un errore, mentre la riga 2 funzionerà perfettamente.

```
1.     bpy.context.object = obj
2.     bpy.context.scene.objects.active = obj
```

Gli attributi di contesto cambiano a seconda di dove vi si accede. La vista 3D ha parti differenti del contesto rispetto alla console, quindi bisogna fare attenzione quando si accede agli attributi del contesto e quando lo stato dell'utente è noto.

III.2.4 OPERATORI

Gli operatori sono strumenti che possono essere ricercati dall'utente dalla vista 3d o attivati tramite pulsanti, voci di menu o key shortcut. Sono l'elemento portante dell'architettura. Tutti i pulsanti dell'interfaccia grafica di Blender chiamano un operatore. Un operatore di solito definisce una funzionalità o l'esecuzione di un set di istruzioni Python che compiono una determinata operazione. Alcuni possono anche contenere un interfaccia grafica a se stante. Di solito utilizzano delle proprietà di un oggetto settate lato utente da interfaccia grafica. Dal punto di vista dell'utente sono dunque uno strumento operativo, dal punto della console Python possono essere eseguiti attraverso il modulo bpy.ops.

```
>>> Bpy.ops.mesh.flip_normals ()
{'FINISHED'}
>>> Bpy.ops.mesh.hide (unselected = False)
{'FINISHED'}
>>> Bpy.ops.object.scale_apply ()
```

Nella sezione Esempio di script vedremo in dettaglio l'uso degli operatori, la loro definizione e del codice di esempio. Teniamo conto che per la stesura di un add-on è fondamentale l'uso degli operatori.

III.2.4.1 POLL NEGLI OPERATORI

Molti operatori hanno una funzione di "poll" che è in grado di controllare che il mouse sia in zona valida o che l'oggetto sia in modalità corretta (Edit Mode, Weight Paint ecc). Quando la funzione di un operatore di tipo pool fallisce, in python viene scatenata un'eccezione.

```
>>bpy.ops.view3d.render_border ()
RuntimeError: Operator bpy.ops.view3d.render_border.poll () failed ,context is incorrect
```

In questo caso il contesto avrebbe dovuto essere la vista 3d con una telecamera attiva. Per evitare di usare la clausola try / except è possibile chiamare le funzioni poll prima di richiamare il relativo operatore, per verificare se è possibile eseguirlo nel contesto attuale.

```
3.      if(bpy.ops.view3d.render_border.poll():
4.      bpy.ops.view3d.render_border()
```

III.2.4.2 TIPI NATIVI

In casi semplici, che restituiscono un numero o una stringa, definire un tipo personalizzato sarebbe scomodo oltre che inuile, per cui queste tipologie di dato sono accessibili come tipi nativi di python.

- Blender float / int / booleano : float / int / booleano.
- Blender enumerator : string (a singolo apice).
- Blender enumerator (multiple) : set di stringhe.

III.2.4.3 TIPI INTERNI

Usati da dataBlocks e collections: bpy.types . bpy_struct . Per dati che contengono gruppi di attributi/meshes/bones/scenes.

III.2.4.4 TIPI MATHUTILS

Usati per vettori, funzioni di Eulero, tipi di matrice e di colore, accessibili da mathutils. Alcuni attributi come bpy.types.Object.location, bpy.types.PoseBone.rotation_euler e bpy.types.Scene.cursor_location possono essere utilizzati come tipi matematici e possono essere utilizzati come insiemi e manipolati.

III.2.4.5 INTERAGIRE CON I MODIFIER

La libreria bpy fornisce un metodo per aggiungere uno dei modificatori ad un oggetto :

```
5.    bpy.ops.object.modifier_add(type='SUBSURF')
```

in cui type è un elemento dell'insieme di simboli di enumerazione che rappresentano ogni modificatore presente in Blender, e cioè:

Type (*enum in ['UV_PROJECT', 'VERTEX_WEIGHT_EDIT', 'VERTEX_WEIGHT_MIX', 'VERTEX_WEIGHT_PROXIMITY', 'ARRAY', 'BEVEL', 'BOOLEAN', 'BUILD', 'DECIMATE', 'EDGE_SPLIT', 'MASK', 'MIRROR', 'MULTIRES', 'REMESH', 'SCREW', 'SKIN', 'SOLIDIFY', 'SUBSURF', 'ARMATURE', 'CAST', 'CURVE', 'DISPLACE', 'HOOK', 'LATTICE', 'MESH_DEFORM', 'SHRINKWRAP', 'SIMPLE_DEFORM', 'SMOOTH', 'WARP', 'WAVE', 'CLOTH', 'COLLISION', 'DYNAMIC_PAINT', 'EXPLODE', 'FLUID_SIMULATION', 'OCEAN', 'PARTICLE_INSTANCE', 'PARTICLE_SYSTEM', 'SMOKE', 'SOFT_BODY', 'SURFACE']*)

la libreria fornisce anche i metodi e le strutture per settare le proprietà dei modifier, come se si stesse interagendo da GUI. Compreso l'applicare il modificatore all'oggetto. Ricordiamo che questo metodo è molto importante perché modifica la topologia della mesh

```
1.  bpy.ops.objects[“mesh_001”].modifier[“Lattice”].object=bpy.data.objects
    [“cage_lattice”]
```

Nell'esempio di codice riportato c'è un esempio di utilizzo delle API per interagire con un operatore lattice (Sezione III.1.2.2). Intanto si suppone che all'oggetto chiamato "mesh_001" sia stato aggiunto un lattice modifier. In questo tipo di modificatore la proprietà object identifica l'oggetto lattice che altro non è che il reticolo che sarà associato all'oggetto `mesh.001`.

III.2.4.6 PROPRIETÀ

Ci sono due diversi modi per definire le proprietà personalizzate agli oggetti o alla scena in Blender tramite Python :

- definire una Property
- definire una ID – Property

III.2.4.6.1 PROPERTY

Di seguito la descrizione delle possibili proprietà di questa tipologia: le proprietà base e le proprietà pesonalizzate. Le proprietà base sono le quelle che troviamo già implementate in Blender.Essendo proprietà di default del sistema potrebbero avere delle limitazioni in acceso. Ad esempio la proprietà di posizione di un oggetto, che è il vettore contenente informazioni sulla posizione dell'oggetto, può essere sia letta che scritta. Altre proprietà , come ad esempio quelle relative agli utenti, sono in sola lettura . Le proprietà personalizzate sono invece di pertinenza dello sviluppatore Pyhton in blender. Quando si vuole aggiungere una nuova proprietà la prima cosa da decidere è la sua tipologia. L'ambiente ci mette a disposizione i tipo base (booleano, float, string, ecc.) ed altre due tipologie particolarmente interessanti: CollectionProperty e le EnumProperty. Le prime

definiscono un insieme di Property, le seconde definiscono un' enumerazione. Di seguito vediamo un esempio di dichiarazione di una proprietà:

```
bpy.types.Object.myProperty = bpy.props.StringProperty ( )
```

Notiamo subito che prima di "myproperty" c'è una dicitura le cui sottoparti sono separate da punti. Questa parte sta ad indicare a che tipo di struttura dell'ambiente la proprietà dovrà essere agganciata. E' possibile anche dichiarare proprietà che si collegano direttamente al contesto. La proprietà `myProperty` sarà dunque creata su qualsiasi oggetto, sia quelli presenti in scena sia in quelli che saranno istanziati successivamente. Una volta aggiunta la proprietà è possibile interagirci, proprio come si trattasse di una proprietà base. Una cosa da tenere a mente è che dopo aver salvato e ricaricato il file Blender la nuova proprietà è stata persa. Qualsiasi oggetto che aveva un valore assegnato alla proprietà, ora avrà ancora a disposizione la proprietà, ma solo come ID - Property. Per rendere i valori che si assegnano in una sessione precedente di nuovo disponibile , è necessario ridefinire la proprietà. Questo si occuperà anche di riassegnare i valori per la proprietà che le erano stati assegnati in una sessione precedente (saranno recuperati dalla ID- property corrispondente).

III.2.4.6.1.1 PROPERTY DINAMICHE PERSONALIZZATE

Il precedente metodo funziona bene per i casi in cui si desidera memorizzare le informazioni statiche su alcuni blocchi dati, ma non sarà di grande aiuto quando si desidera che le proprietà siano dinamiche. In questo caso è necessario assegnare una funzione alla proprietà. Il modo per farlo è quello di utilizzare la funzione integrata di Python property(). Un'ulteriore differenza è che le proprietà personalizzate e dinamiche non vengono memorizzati durante il salvataggio di un file .blend. Nemmeno come ID - propriety. Per avere le nostre proprietà disponibili in diversi file .blend (anche dopo aver salvato e ricaricato), bisogna aggiungere la vostra proprietà in .. / .blender / scripts / modules /bpy_types.py (dove ../ è la cartella in cui blender è installato).

III.2.4.6.2 ID – PROPERTY

Le ID- Property sono le proprietà che vengono assegnate ai singoli blocchi dati . Non su qualsiasi datablock, ma solo su quelli che sono una sottoclasse di ID- type . Esempi sono Lamp,Meshes Objects e WindowManagers .La creazione di una nuova ID –property è molto semplice :

```
1.      ob=bpy.context.active_object    #ottenere l'oggetto attivo
2.      ob[ "foo"]= "bar" #ID-Property create e assegnata
3.
4.      print(ob["foo"]) # stampa la stringa "bar"
5.      #print(ob.foo)   # non funziona , foo non è un attributo
6.      print(ob.items()) # stampa tutte le ID-Property di ob
```

Le ID-property sono memorizzate sul data block come se quest'ultimo fosse un dictionary , vale a dire che la proprietà viene associata al'oggetto con un etichetta che ne definisce il nome. E ogni qual volta che si vuole accedere alla proprietà si interrogherà l'oggetto con l'uso di quadre, in cui passeremo il valore di chiave del dizionario. Nel caso in esempio sarà la stringa "foo".

III.2.4.6.3 RIFERIMENTI

Le ID- Property hanno il vantaggio di poter essere memorizzate in un file.blend e sono facili da monitorare perché di default sono mostrati nell'interfaccia utente . Essi appaiono nel pannello Custom Property . Quindi, se si aggiunge un ID-Property di un oggetto apparirà nel pannello delle Proprietà personalizzate relative all'oggetto. Questo rende molto facile cambiare i loro valori, anche per gli utenti normali, il pannello può anche essere usato per aggiungere nuove ID-Property senza dover utilizzare Pyhton. Per visualizzare l' ID–property utilizzando `UILayout.prop( ),` non si può semplicemente passare " foo ", come se fosse una property, ma bisogna usare la stessa convezione usata in creazione, dunque [" foo "] .

```
1.      class  myPanel(bpy.types.Panel) # pannello per rendere visibile la proprietà
2.        bl_space_type=”VIEW_3D” # compare nella finestra 3d
3.        bl_region_type=” UI ” # compare in pannello delle proprietà
4.        bl_label= “ My Panel” #nome del nuovo pannello
5.
6.        def draw(self,context):
7.          self.layout.prop(bpy.context.active_object,’[“foo”]’)
8.
9.      bpy.utils.register_class(myPanel) #registro il pannello
```

III.2.5 ESEMPI DI SCRIPT

In questo esempio creeremo uno script che, dato un oggetto in scena chiamato “Icosphere”, crea una nuova mesh che deriva dall’originale ma a cui sono stati sottratti i vertici che sono lontani dal centro dell’oggetto, rispetto all’asse z, più di un certo valore.
Iniziamo col prendere dal contesto gli oggetti presenti e prelevare l’oggetto di nome “Icosphere”. Per essere sicuri che tutto vada a buon fine passiamo alla modalità OBJECT, deselezioniamo tutti gli oggetti in scena, selezioniamo via codice l’oggetto prescelto e lo rendiamo l’oggetto attivo in scena. Devo renderlo attivo perché in modalità Edit si lavora solo sull’oggetto attivo in scena.

```
1.  bpy.ops.object.mode_set ( mode=’OBJECT’)
2.
3.  #deseleziono tutto
4.  bpy.ops.object.select_all ( action=’DESELECT’)
5.  is_from.select=True
6.
7.  #attivo l’oggetto nella scena
8.  bpy.contex.scene.object.active=is_from
```

Ora attiviamo la modalità EDIT, procedendo col creare una `bmesh` dal `.data` dell’oggetto selezionato , così facendo posso andare ad interagire sui vertici,spigoli e facce della mesh.

```
9.  bpy.ops.object.mode_set ( mode=’EDIT’)
10. mesh=bmesh.from_edit_mesh( is_from.data)
```

Seguendo un semplice algoritmo troveremo i vertici a coordinata massima e minima rispetto all'asse z, calcoleremo il centro rispetto alla stessa coordinata ed al valore di scarto, cioè una porzione della distanza tra il valore della coordinata z nel centro e il valore della stessa coordinata nel suo punto minimo.

```
#cerco min e max rispetto all'asse z
minz=mesh.verts[0].co[2]
maxz=mesh.verts[0].co[2]

for ver in mesh.verts:
    if ver.co[2] < minz :
        minz = ver.co[2]
    if ver.co[2] > maxz :
        maxz=ver.co[2]

middle= (maxz + minz)/2
scarto=(middle – minz )/percent_val
```

Copiamo la mesh e, ciclando sui suoi vertici, elimino quelli che hanno coordinata z esterna all'intervallo [centro-scarto,centro+scarto]

```
mesh2=mesh.copy()
for vertice in mesh2.verts:
    if(vertice.co[2]>middle+scarto)or(vertice.co[2]<middle-scarto):
        Mesh2.verts.remove(vertice)
```

Creiamo poi una nuova mesh e ci associamo l'oggeto bmesh modificato

```
me=bpy.data.meshes.new("Mesh")
mesh2.to_mesh(me)
mesh2.free()
```

Aggiungiamo la mesh alla scena, creando un nuovo oggetto. Associamo poi al nuovo oggetto il fattore di scala presente nell'originale.

```
31. Scene=bpy.context.scene
32. Object_mesh_to=bpy.data.objects.new(“Object”,me)
33.
34. vector_scale=c.scale
35. object_mesh_to.scale=vector_scale
36.
37. scene.objects.link(object_mesh)
```

In figura si può vedere il risultato di questo script per diversi fattori di scarto.

.

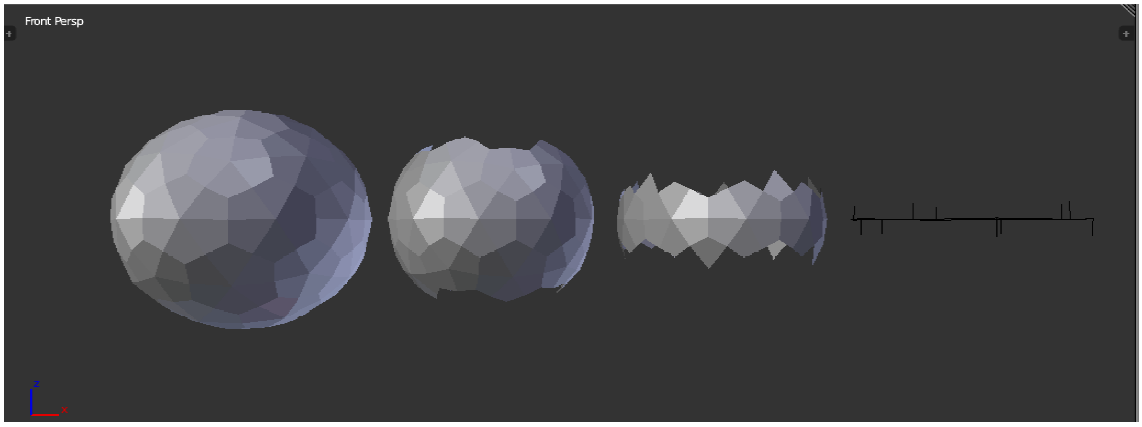

FIGURA 11 - RISULTATI CON DIVERSI FATTORI DI SCARTO

III.2.5.1 ESEMPIO DI OPERATORE BASE

Per la stesura di questa sezione abbiamo seguito i passi indicati in (16).Partiamo dalle prime due variabili che si incontrano andando ad osservare un file di scripting contenente un operatore, che sono:

- **bl_id name**, che fornisce un nome interno per il nostro operatore
- **bl_label**, che fornisce un’etichetta che sarà usata per identificare un operatore, ad esempio all’interno di un menu.

Il prefisso “bl_” sta ad indicare che le due variabili di classe sono specificate perché richieste da Blender e non sono variabili locali definite dal programmatore per scopi interni all’operatore. Da notare che, per convenzione, sceglieremo il valore di bl_idname in modo che sia una stringa che inizia con il valore “object”. Questa è una convenzione che

identifica questo operatore come uno di quelli che vanno ad agire su un istanza di un oggetto Blender .

```
1.    import bpy
2.    class MoveOperator(bpy.types.Operator):
3.        bl_idname = "object.move_operator"
4.        bl_label = "Move Operator"
5.        def execute(self, context):
6.            context.active_object.location.x += 1.0
7.            return {'FINISHED'}
8.
9.    def register():
10.       bpy.utils.register_class(MoveOperator)
11.
12.   if __name__ == "__main__":
13.       register()
```

FIGURA 12 - CODICE DELL'OPERATORE MOVEOPERATOR(16)

Prendiamo l'esempio dell'operatore in figura 12 e osserviamo che l'elemento successivo alla definizione dell'etichetta e del nome che lo identifica è la dichiarazione del metodo `execute()`, che viene chiamato quando l'utente richiama l'operatore.

Nei suoi argomenti viene passato `context`, che contiene le informazioni sul contesto in cui l'operatore è stato invocato. Il contesto registra efficacemente lo stato di Blender, per esempio qual'é l'oggetto attivo, in che modalità siamo (Object o Edit), la scena e l'insieme degli oggetti. Il metodo, partendo dal riferimento dell'oggetto attivo prelevato dal contesto (`context.active_object`) ne incrementa la posizione sull'asse x di un'unità. Per segnalare che la funzione è stata conclusa restituisce un set a singolo elemento contenente una singola stringa `'FINISHED'`.

Dichiarata la classe, dobbiamo ancora dire a Blender di rendere il nostro operatore disponibile all'utente. Per questa di questo di occupa il metodo `register`(). Le righe finali servono ad assicurarsi che se eseguiamo lo script dal Text Editor cliccando su "Run Script " (presente sulla barra in basso nella relativa vista), riusciremo a registrare l'operatore. La registrazione poteva anche essere fatta direttamente chiamando la funzione `register_class()`, fornendola a livello di file però, facciamo in modo che la nostra classe operatore sia registrata automaticamente, se installeremo il nostro script come una add-on.

```
6.     import bpy
7.     import os
8.
9.     filename = os.path.join(os.path.dirname(bpy.data.filepath), "addon-
move3.py")
10.    exec(compile(open(filename).read(), filename, 'exec'))
```

FIGURA 13 - SCRIPT PER ESEGUIRE FILE PYTHON ESTERNI

Il file di scripting potrà essere eseguito direttamente in qualsiasi directory dichiarata o dalla stessa directory in cui è memorizzato il file .blend. Il codice in figura 13 fa proprio questo, cioè attiva lo script "addon-move3.py" dalla stessa directory in cui si trova il file .blend. Chiaramente sarebbe molto meglio aggiungere il nostro operatore ad un menu vero, ad esempio nel menu Object della View3d. Tutti i menu che troviamo in Blender sono stati definiti all'interno di uno script, infatti passandoci sopra il mouse ogni elemento della GUI rivela il nome del file python associato

III.2.5.1.1 AGGIUNGERE UN OPERATORE IN UN MENU

Per renderizzare il nostro operatore all'interno del menu, dobbiamo aggiungerlo. Questo si fa, come abbiamo già spiegato, con la funzione `register()`. L'argomento add_object_button è una funzione che prende i due parametri `context` e `self`, quest'ultimo si riferisce al menu che è stato visualizzato. Si aggiunge l'operatore al layout del menu chiamando la funzione `operator()` di questo layout. I tre argomenti passati al nostro operatore sono il nome interno dell'operatore, il testo che dovrà apparire nel menu e l'icona da utilizzare. Il docstring (il testo circondato da tre virgolette ('") all'inizio della definizione della classe, apparirà come "hover text" quando passeremo con il mouse su un elemento di menu.

III.2.5.1.2 AGGIUNGERE PROPRIETÀ AD UN OPERATORE

Molti operatori cambiano il loro comportamento in base alle proprietà impostate dall'utente e noi vorremmo che il nostro operatore faccia altrettanto. Proviamo quindi ad esempio a definire due proprietà : distanza e direzione e facciamo in modo che il nostro `execute()` utilizzi questi valori. La registrazione di un operatore non è sufficiente a farlo apparire

nella barra degli strumenti dopo che il nostro operatore è stato selezionato dal menu. Per farlo dobbiamo impostare `bl_options` con un insieme che contenga il valore 'REGISTER', possiamo inserire anche la voce `'UNDO'` per fornire all'utente la possibilità di rendere nulle le modifche ai valori di default. Le nostre proprietà sono definite come variabili di classe che saranno inizializzate utilizzando le opportune proprietà di default. Blender ne ha tutta una serie, documentata per il modulo bpy.props (III.2.4.6).

```
…

1.      from bpy.types import Operator
2.      from bpy.props import FloatVectorProperty, FloatProperty
3.
4.      class Move3Operator(bpy.types.Operator):
5.       """Move3 Operator"""
6.       bl_idname = "object.move3_operator"
7.       bl_label = "Move3 Operator"
8.       bl_options = {'REGISTER', 'UNDO'}
9.
10.      direction = FloatVectorProperty(
11.        name="direction",
12.        default=(1.0, 1.0, 1.0),
13.        subtype='XYZ',
14.        description="move direction"
15.        )
16.
17.      distance = FloatProperty(
18.        name="distance",
19.        default=1.0,
20.        subtype='DISTANCE',
21.        unit='LENGTH',
22.        description="distance"
23.        )
24.
25.      def execute(self, context):
26.       dir = self.direction.normalized()
27.       context.active_object.location += self.distance * dir
28.       return {'FINISHED'}
29.
30.      @classmethod
31.      def poll(cls, context):
32.       ob = context.active_object
33.       return ob is not None and ob.mode == 'OBJECT'
34.     …
```

FIGURA 14 - ESEMPIO DI SCRIPT CON PROPERTY

Prendiamo l'esempio in figura 14. E' stata impostata la presa la direzione come `FloatVectorProperty`, che è un vettore con componenti x,y,z,. In realtà `FloatVectorProperty` offre già un sottotipo di distanza usato anche per definire delle normali, ma nell'esempio si è voluto utilizzare i sottotipo 'XYZ'. Definiamo quindi la distanza come `FoatProperty` e settiamo i sui attributi: un nome, un valore di default e

dei valori di massimo e minimo. Ci vogliono anche argomenti di unità e sottotipo che non cambieranno il loro valore ma altereranno la visualizzazione del widget e aggiungeranno le unità appropriate. Il nuovo execute recupererà la distanza e la direzione e dopo aver normalizzato la direzione cambia la posizione dell'oggetto attivo. Tutto questo elimina ogni sforzo per definire un'interfaccia utente, chiaramente è possibile anche disegnare un layout custom fornendo una funzione draw(), ma per operatori semplici questo non è quasi mai necessario. L'ultima cosa da tener presente è la funzione poll(). Questo metodo di classe ci fornisce un modo per controllare se l'operatore è applicabile. Qui l'abbiamo definito per verificare se c'è un oggetto attivo in tutti e in caso affermativo, se è in modalità oggetto. Se restituisce false, qualsiasi voce di menu per questo operatore non è selezionabile.

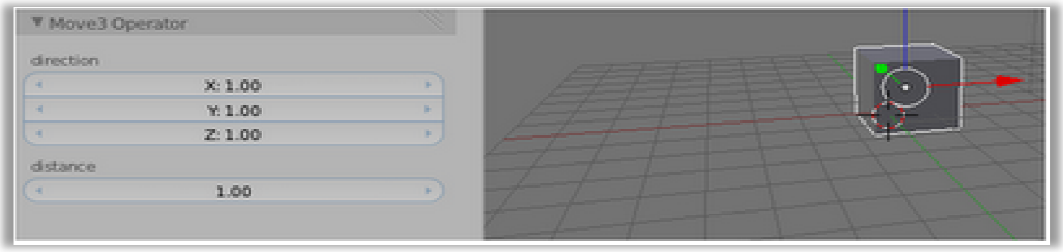

FIGURA 15 - ADDON IN USER INTERFACE(16)

III.2.5.1.3 DISTRIBUZIONE DI UN ADD-ON

Fino a ora abbiamo eseguito il nostro script dall'editor di testo, ma normalmente noi vorremmo che lo script sia disponibile per tutti i .blend file e l'utente possa scegliere di attivare l'add-on dale user-preferences. Per fornire all'utente e a Blender informazioni rilevanti sulla nostra Addon aggiungeremo alcune informazioni extra all'inizio del nostro codice come mostrato in seguito in figura 16 e salviamo il nostro script nella directory add-ons, infine riavviamo blender.

```
bl_info = {    "name": "Example For Object",  "author": " Fabio Caputo",
               "version": (1, 0),   "blender": (2, 6, 7),
               "location": "View3D > Object > ExampleForObject",
               "description" : "Muovere l'Object attivo",
               "warning": "",   "wiki_url": "", "tracker_url": "",
                "category": "Object"}
```

FIGURA 16 - INFORMAZIONI SULL'ADDON

Gli addons salvati nella directory addons finiranno tra gli Adddon. Possiamo gestire sia l’attivazione che la disattivazione di un addon dalla pagina di gestione, cioè dal menu della barra info selezionare File / UserPreferences / Addons. Nell’interfaccia sono presenti tutti gli addons salvati e attivabili.

IV. REALIZZAZIONE DELL'ADD-ON PER BLENDER

A supporto della nostra implementazione è stato utilizzato un codice preesistente sviluppato in linguaggio c per la realizzazione di deformazioni surface-based tramite operatore laplaciano. L'adattamento a Blender ha portato ad una riscrittura quasi completa del codice. Riportiamo dunque la base teorica per l'implementazione e di seguito, riportiamo le scelte implementative, gli schemi di definizione degli operatori e degli algoritmi realizzati, la definizione del pannello per iteragire con l'Add-on, le strutture e le funzioni di utility principali che sono state introdotte.

IV.1 RAPPRESENTAZIONE PARAMETRICA E OPERATORE LAPLACIANO

In questo paragrafo affrontiamo il problema di trovare la X: $\Omega \epsilon \mathbb{R}^2 \rightarrow \mathcal{M} \epsilon \mathbb{R}^3$ che definisce la parametrizzazione della nostra mesh. Il modo di rappresentare la mesh è una scelta implementativa che condiziona fortemente il tipo di proprietà geometriche e combinatorie dell'oggetto, ancorché la qualità percepita. Ad esempio la rappresentazione a facce triangolari, quindi lineare a tratti, fornisce un modo immediato di rappresentare la mesh e di conoscere le sue proprietà topologiche. Inoltre si può arrivare anche ad approssimare le proprietà differenziali di una superficie continua. Dall'altro lato però possono rendere difficili alcune attività di modellazione. Utilizzeremo un metodo che si basa su operatori lineari sulla mesh che ne definiscono una rappresentazione differenziale. Rispetto alle coordinate cartesiane globali che danno solo la posizione spaziale di ogni punto, una rappresentazione differenziale porta con se informazioni sulla forma locale della superficie e sui dettagli locali, in dimensione e orientamento. Quindi definire delle operazioni sulla superficie che cercano di mantenere inalterate queste informazioni, diventano operazioni con conservazione di dettaglio. Sia $M(V, E, F)$ una Mesh a facce triangolari con n vertici, in cui V denoterà i vertici, E l'insieme degli edge e F l'insieme delle facce. Per convenzione diciamo che ogni vertice della mesh è rappresentato tramite coordinate cartesiane globali nella forma $v_i = (x_i, y_i, z_i$. Ad esempio potremmo definire il differenziale o le coordinate δ di v_i come la differenza delle coordinate assolute di v_i con il

centro i massa dei sui vicini immediati nella mesh come descritto in(17). La trasformazione del vettore dalle coordinate cartesiane globali al vettore di coordinate δ può essere rappresentato in forma matriciale. La matrice L è chiamata topologia (o grafo) Laplaciano della mesh. Dal punto di vista della geometria differenziale nell'esempio citato le coordinate δ possono essere viste come la discretizzazione dell'operatore continuo di Laplace-Beltrami , se assumiamo che la nostra mesh sia un'approssimazione lineare a tratti di una superficie continua. Questa discretizzazione geometrica del Laplaciano non è l'unica possibile anzi ne esistono i qualità migliore. Di seguito sarà riportato un breve escursus a livello implementativo dei diversi tipi di discretizzazione geometrica del Laplaciano.

IV.1.1 DISCRETIZZAZIONE TRAMITE VALENZA E DISTANZE

Una delle possibilità di discretizzazione è l'utilizzo di un metodo che non tiene conto in maniera forte della geometria della superficie, ma si limita a dare dei pesi in maniera semi o totalmente uniforme. In questo tipologia ricadono le seguenti forme, dove per tutti vale che vi è il vertice di riferimento, $N(i) = \{ j \mid (i,j) \epsilon E \}$ e L è la matrice che rappresenta la topologia Laplaciana della Mesh.

- Constant, in questo caso il peso associato ad ogni vertice vj collegato a vi è unitario.

$$Lij = \begin{cases} -d_i & i = j \\ 1 & (i,j) \in E \\ 0 & altrimenti \end{cases} \text{ , in cui } d_i = |\, N(i) \,| \qquad (42)$$

- Fujiwara , il peso associato ad ogni vertice è calcolato rispetto alla distanza tra vi e i suoi vicini.

$$Lij = \begin{cases} -\frac{2}{S^2} & i = j \\ \frac{2}{S * sj} & (i,j) \in E \\ 0 & altrimenti \end{cases} \text{ , in cui } s_i = \left\| v_j - v_i \right\| \text{ e } S = \sum_{j \epsilon N(i)} s_j \qquad (43)$$

- Umbrella, in questo caso il peso associato ad ogni vertice vj collegato a vi deriva dalla sua valenza d_i.

$$Lij = \begin{cases} -1 & i = j \\ \frac{1}{d_i} & (i,j) \in E \\ 0 & altrimenti \end{cases} \quad \text{, in cui } d_i = |N(i)| \tag{44}$$

IV.1.2 DISCRETIZZAZIONE TRAMITE COTANGENTE E VALOR MEDIO

Ci sono poi, oltre alle precedenti , tipologie con maggiore qualità di approssimazione della discretizzazione geometrica del Laplaciano. Nell'add-on sono state introdotte due delle tipologie di discretizzazione più famose. Entrambe tengono conto delle proprietà geometriche locali tramite gli angoli che si formano tra gli edge uscenti dal vertice di riferimento vi. Prima di tutto abbiamo bisogno di conoscere, per ogni vertice vi, non solo i vertici opposti appartenenti a tutti gli edge collegati a vi, ma ne abbiamo bisogno anche in forma ordinata. A questo scopo utilizzeremo un algoritmo che ciclando in un'unica direzione restituisce tre dizionari ordinati contenenti rispettivamente i vertici, le facce e gli edge.

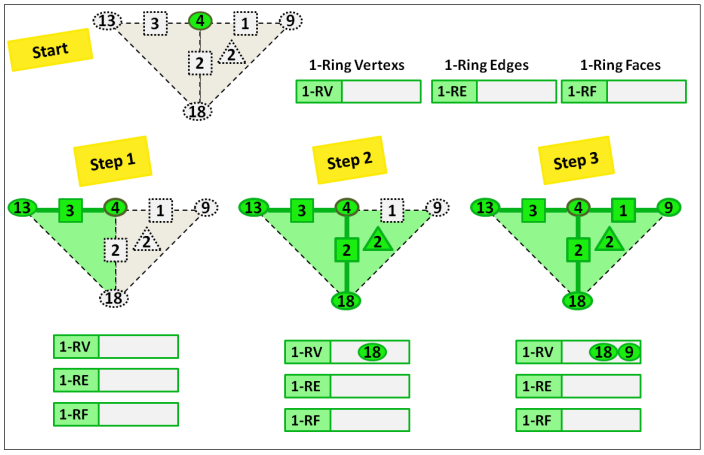

FIGURA 17 ALGORITMO 1-RING

L'utilità di avere una serie di vertici ordinati è fondamentale nelle forme di discretizzazione del Laplaciano che tengono della conformazione geometrica dalla mesh. In più dopo la risoluzione del sistema non potremmo più avvantaggiarci delle strutture dato

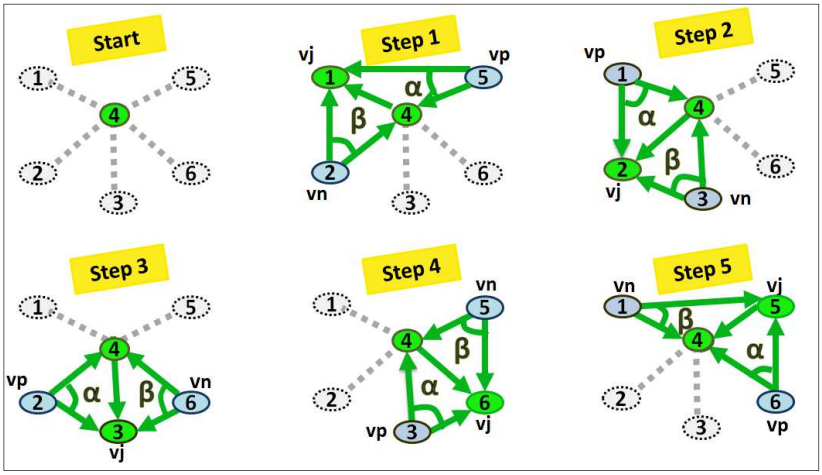

FIGURA 18 – ALGORITMO DI RILEVAZIONE DEGLI ANGOLI PER IL CALCOLO DEI PESI

Nell'esempio in figura 18 possiamo osservare l'algoritmo che riconosce gli angoli necessari per il calcolo di entrambe le discretizzazioni. Vediamo come per ogni vertice vj $\epsilon N(i)$ si possono identificare facilmente il suo precedessore vp ed il suo sucessore vn. L'algoritmo sfrutta il metodo per ricavare l' 1-Ring che, come descritto precedentemente restituisce i vertici in maniera ordinata secondo una stessa direzione. Una volta riconosciuti i due vertici è immediato ricavare l'edge ed il relativo angolo. Chiameremo α l'angolo compreso tra i vettori vpj e vpi e chiameremo β l'angolo compreso tra i vettori vnj e vni .
Di seguito le formulazioni delle due tipologie, dando per buone le convenzioni enunciate nella precedente descrizione:

- Cotangente, in questo caso si utilizza l'area di Voroni di vi e i due angoli α e β appena descritti. Deriva dalla formulazione data in (2):

$$\delta_i^c = \frac{1}{|\Omega_i|} \sum_{j \epsilon N(i)} \frac{1}{2} \left(cot\alpha_{ij} + cot\beta_{ij}\right)(vi - vj) \qquad (45)$$

In cui $|\Omega_i|$ è l'area di Voronoi e con α_{ij} e β_{ij}si intendono gli angoli α e β opposti all'edge (i , j). Se ne deduce che la rispettiva matrice L sarà :

$$Lij = \begin{cases} -\sum_{j\epsilon\, N(i)} w_{ij} & i = j \\ w_{ij} = \frac{1}{2|\Omega_i|}\left(cot\alpha_{ij} + cot\beta_{ij}\right) & (i,j) \in E \\ 0 & altrimenti \end{cases} . \qquad (46)$$

Purtroppo questo tipo di discretizzazione può generare dei pesi negativi che possono dare problemi per angoli molto grandi , per via delle proprietà delle cotangenti vicino a π.

- Mean Value , per ottemperare a i problemi che potrebbero provocare pesi negativi la formulazione a valor medio persegue l'intento di emulare quella a cotangente ma con soli pesi positivi perché di tipo convesso. La matrice L sarà così definita:

$$Lij = \begin{cases} -\sum_{j\epsilon\, N(i)} w_{ij} & i = j \\ w_{ij} = \frac{\tan\left(\frac{\theta_{ij}^1}{2}\right)+\tan\left(\frac{\theta_{ij}^2}{2}\right)}{\|vi-vj\|} & (i,j) \in E \\ 0 & altrimenti \end{cases} . \qquad (47)$$

- Cotangente senza Area, questa formulazione è identica a quella relativa alla cotangente senza l'utilizzo della Voronoi Area. Da cui L sarà:

$$\text{Lij}=\begin{cases} -\sum_{j\epsilon\, N(i)} w_{ij} & i = j \\ w_{ij} = \left(cot\alpha_{ij} + cot\beta_{ij}\right) & (\,i,j\,) \in E \\ 0 & altrimenti \end{cases} . \qquad (48)$$

Nella precedenti formulazioni compaiono θ_{ij}^1 e θ_{ij}^2 , altro non sono che gli angoli compresi rispettivamente tra i vettori vpi e vij e tra i vettori vni e vij (vedi figura 19).

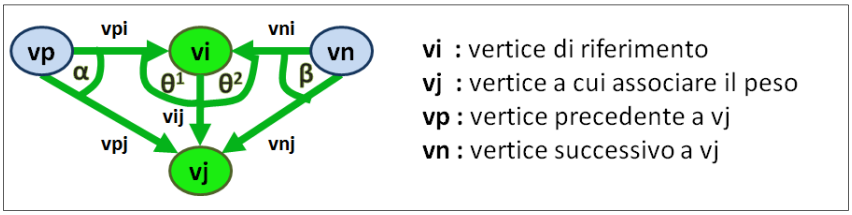

FIGURA 19 ANGOLI DESCRITTI

IV.2 MODELLO RISOLUTIVO

L'idea di risoluzione si basa sul fissare delle aree F ∈ $\mathcal{M}$ e definire uno spostamento per aree così dette handle $\mathcal{H} \in \mathcal{M}$. Quindi $\mathcal{M}$ deve essere ricalcolato di volta in volta risolvendo il problema differenziale quando l'utente manipola i vincoli di contorno, cioè manipola le aree $\mathcal{H}$. Considerando l'energia di deformazione e i vincoli lineari e non lineari rappresentati da un generico Φ() , la deformazione vincolata può essere rappresentata come:

$$min_{X'} E(X' - X) \quad soggetto\ a\ \Phi(X') \tag{49}$$

Siamo interessati a valutare il problema variazionale vincolato che coinvolge l'energia di curvatura E(T) per superfici discrete, cioè mesh tirangolari M, che rappresentano un'approssimazione lineare a tratti delle superfici continue $\mathcal{M}$. All'interno della Mesh M possiamo distinguere tre insiemi di vertici rispetto al loro ruolo nella deformazione:

- Handle ={ vj | vj spostato da interfaccia utente, vj ϵ M}
- Fixed = { vj | vj resta fisso dopo la deformazione , vj ϵ M}
- Free = { vj | vj spostato durante la deformazione , vj ϵ M}

in cui Handle e Fixed sono i corrispondenti discreti di F $e\ \mathcal{H}$. I vincoli posizionali da imporre possono essere incorporati nel sistema di risoluzione in modalità Hard o Soft. Nella modalità Hard si suppone che i vertici appartenenti all'insieme Handle restino nella posizione manipolata lato user interface. Dunque questo tipo di vincoli sono da preferire negli strumenti di editing classici in cui deve essere raggiunta la posizione esatta. I vincoli soft non costringono all'esatta posizione finale i vertici manipolati , questo tipo di vincoli sono preferibili in uno strumento sketch-based poiché si consente all'utente di inserire posizioni imprecise e suggerire la forma desiderata, senza averla indicata puntualmente. Diciamo che M è definita da un insieme T di triangoli $Ti\ , i = 1..N_t$ che copre interamente M e da un insieme X di vertici $Xi, i = 1..N_v$ dove $Xi \epsilon \mathbb{R}^3$ e l'i-esimo vertice è $X_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, con un vettore di norma n_i associato. Introduciamo quindi gli ingredienti necessari alla deformazione variazionale proposta , la cui formulazione nel caso continuo era

$$E_T(\mathcal{M}) = \int_{\mathcal{M}} (H^2 - 2K)\, d\mathcal{M} \ . \tag{50}$$

Visto che per una superficie liscia $\Delta_{\mathcal{M}} = 2Hn$ è un'approssimazione discreta del vettore di curvatura media $H_i n_i$ associata al vertice Xi , può essere derivata usando la seguente forma discreta:

$$H_i n_i = L(X_i) = \frac{1}{2A_i}\sum_{j\in N(i)} w_{ij}\,(X_j - X_i) \tag{51}$$

Dove L rappresenta la rappresentazione dell'operatore locale di Laplace-Beltrami $\Delta_{\mathcal{M}}$ su M, N(i) è l'insieme dei vicini 1-Ring del vertice X_i, A_i è l'area di Voronoi attorno a X_i e i pesi w_{ij} sono i numeri positivi che soddisfano a condizione di normalizzazione $\sum_{j\in N(i)} w_{ij} = 1$, useremo di default la discretizzazione a cotangente senza area, descritta nella sezione precedente. La nozione della curvatura Gaussiana può essere estesa anche alla superficie discreta M ed è supportata sui vertici $X_i \epsilon\, X$. Infatti perché il teorema di Gauss-Bonnet sia verificato, dobbiamo prendere

$$\int_{\mathcal{M}} K\ d\mathcal{M} \coloneqq \sum_i K_i\ , K_i\ = \frac{1}{A_i}(2\pi - \sum_{j\epsilon N(i)} \theta_j) \tag{52}$$

Dove con θ_i intendiamo gli angoli interni incidenti a X_i .Questa forma rappresenta lo standard per mesh triangolari. Considerando allora il vettore di spostamento $d = X' - X$, i modelli di deformazione basati sull'energia per la superficie rappresentata dalla Mesh può essere discretizzato come segue:

$$E_M: \quad Ld = 0 \quad \Rightarrow \quad LX' = LX \tag{53}$$

$$E_B: \quad L^2 d = 0 \quad \Rightarrow \quad L^T LX' = L^T LX \tag{54}$$

$$E_C: \quad k_s\, Ld + k_b L^2 d = 0 \quad \Rightarrow \quad (k_s\, L + k_b L^2)X' = (k_s\, L + k_b L^2)X \tag{55}$$

ed infine il modello ad energia totale:

$$E_T: \quad L^2 d - 2LGd = 0 \quad \Rightarrow \quad (L^2 - 2LG)X' = (L^2 - 2LG)X \tag{56}$$

In cui $G = H^2 - K$.

Ognuno di questi modelli si può portare in forma di un generico sistema lineare Ax=b a partire da una matrice di coefficienti sparsa di dimensioni $N_v x\, N_v$.

IV.2.1.1 VINCOLI NON LINEARI

In questo paragrafo introdurremo in metodo risolutivo per i vincoli non lineari introdotto da (2). Di solito il valutare $\Delta_{\mathcal{M}} H$ su X_i (per X_i interno , cioè $X_i \in X(\mathcal{F}\cup\mathcal{H})$ o un vertice

esterno, che sia $X_i \in \partial\mathcal{M}$) implica vertici vicini 2-Ring da Xi . alcuni di loro possono essere dei vertici interni e gli altri sono vertici esterni. I vertici interni sono trattati come incognite nelle equazioni discretizzate e gli esterni sono incorporati nel lato destro del sistema esposto nella precedente sezione. Un modo per applicare vincoli posizionali soft è quello di incorporarli in una formulazione penalizzante dell'energia funzionale discreta

$$min_{X'} E(X' - X) + \frac{\lambda}{2}\|X' - C\|^2 \tag{57}$$

Per avvicinarsi l'interpolazione dei vincoli C , il parametro λ deve essere scelto grande a sufficienza . Tuttavia il numero di condizioni della matrice cresce con λ , quindi un peso più elevato può causare problemi numerici. Considerando la discretizzazione con l'energia di curvatura Totale, i vincoli posizionali, i vincoli per preservare i dettagli ed infine partendo dal fatto che le $\hat{\delta}(X')$ sono incognite nel processo di deformazione è stata proposta la seguente minimizzazione dell'energia funzionale per risolvere il problema di deformazione mesh:

$$min_{X',\hat{\delta}}\, \mathcal{L}(X', \hat{\delta}) \tag{58}$$

$$\mathcal{L}\left(X', \hat{\delta}\right) \coloneqq E(X' - X) + \frac{\lambda_1}{2}\|X' - C\|^2 + \frac{\lambda_2}{2}\left\|LX' - C\hat{\delta}\right\|^2 \tag{59}$$

Il cui minimo può essere determinato dal alternare una procedura di minimizzazione, cioè, risolveremo per $k = 0, .., num_p$ volte di seguito

$$\hat{\delta}^{(k+1)} = argmin_{\hat{\delta}}\, \mathcal{L}(X'^{(k)}, \hat{\delta}) \tag{60}$$

$$X'^{(k+1)} = argmin_{X'}\, \mathcal{L}(X', \hat{\delta}^{(k+1)}) \tag{61}$$

Poiché $\mathcal{L}\left(X', \hat{\delta}^{(k+1)}\right)$ è continua e differenziabile in X', la soluzione $X'^{(k+1)}$ della seconda minimizzazione si ottiene imponendo:

$$0 = \nabla_{X'}\, \mathcal{L}\left(X', \hat{\delta}^{(k+1)}\right) = \tag{62}$$

$$(L^2 - 2LG)(X' - X) + \lambda_1(X' - C) + \lambda_2\left(L^T\left(LX' - \hat{\delta}^{(k+1)}\right)\right) \tag{63}$$

dove $G| = H^2 - K$.

In forma matrice-vettore , la soluzione per la nuova maglia vertici X ', è data dalla soluzione del sistema sovra determinato

$$\begin{bmatrix} (L^2 - 2LG) \\ 0 \quad \sqrt{\lambda_1} I_n \\ 0 \quad \sqrt{\lambda_2} L^T L \end{bmatrix} X' = \begin{bmatrix} (L^2 - 2LG)X \\ \sqrt{\lambda_1} C \\ \sqrt{\lambda_2} L^T \hat{\delta}^{(k+1)} \end{bmatrix} \tag{64}$$

ove il blocco di $A = (L^2 - 2LG)$ rappresenta l'energia interna , $I_n \epsilon \mathbb{R}^{nxn}$ è la matrice identità che richiede un riordinamento delle righe di L, $C \in \mathbb{R}^n$ è un vettore di elementi c_i per ognuna degli n vincoli posizionali. Il sistema ha dimensioni $(N_v + 2_n)\ x\ N_v$ ed è a rango pieno, questa ha un'unica soluzione ai minimi quadrati. Bisogna implementare un risolutore interattivo che ci permetta di mantenere matrici di grandi dimensioni. Le iterazioni saranno poi terminate non appena la norma della differenza tra le ultime due soluzioni sia inferiore o uguale a 10^{-4}. L' uso della $L = D\bar{L}$ e $K = D\overline{K}$ con la matrice dell'area scalare $D_{ij} = \frac{1}{A_i}$ nel sistema, permette di estendere l'invarianza a trasformazioni rigide e le proprietà di scalatura uniforme dell'energia E_T anche all'impostazione discreta.

Per aggiornare $\hat{\delta}^{(k+1)}$, in prima approssimazione possiamo considerare $\hat{\delta}^{(k+1)} = LX'^{(k)}$. In particolare se consideriamo che, per mantenere i dettagli, la deformazione deve essere ottenuta da $\frac{1}{2}\int_{\mathcal{M}}(\Delta_{\mathcal{M}}X' - \delta)^2\, d\mathcal{M}$ con $\delta = \Delta_{\mathcal{M}}(X')$ ad ogni Step è stata supposta una procedura in due fasi :

- **Fase 1:**

Risolvere su ogni vertice X_i con $N(i)$ vicini , per $\mu^i = \mu^i_1, \mu^i_2, \dots, \mu^i_{N(i)}$:

$$\sum_{j\epsilon N(i)} \mu^i_j((X_j - X_i) \otimes (X_{j-1} - X_i)) = \delta_i \qquad (65)$$

Dove δ_i sono le coordinate laplaciane della prima deformazione e $(X_j - X_i) \otimes (X_{j-1} - X_i)$ è il vettore normale al triangolo X_i , X_j , X_{j-1} su X_i . il sistema lineare sovradimensionato può essere rappresentato come:

$$A_i\mu^i = \delta_i \qquad (66)$$

Dove A ha dimensioni 3 x N(i).

- **Fase 2:**

Inseriamo la μi calcolata in

$$d_i(X') = \sum_{j\epsilon N(i)} \mu^i_j((X_j' - X_i') \otimes (X_{j-1}' - X_i')) = \delta_i \qquad (67)$$

con X' sono i vertici della mesh deformata $\mathcal{M}'$. Fino a che le μ^i sono le medesime, prima e dopo la deformazione , $d_i(X) = T_i\delta_i$ per le rotazioni locali T_i. Infine, le coordinate laplaciane sono normalizzati come segue :

$$\hat{\delta}(X_i) = \frac{d_i}{\|d_i\|}\|\delta_i\| \tag{68}$$

Questo significa che useremo le coordinate laplaciane del precedente passo iterativo come direzione di destinazione, tenendo la lunghezza delle coordinate Laplaciane originali come valore di riferimento. Dato che le indicazioni da conservare sono quelle in posizione di riposo, il primo step può essere eseguito come passo preliminare , mentre il secondo aggiorna le nuove coordinate laplaciane ad ogni step k. Nel caso volessimo forzare i vincoli posizionali hard che impongono i vertici nell'esatta locazione a loro predestinata, setteremo λ_1 =0 quindi il sistema che sostituisce il blocco della matrice A con una ridotta matrice An ∈ R (Nv - n) × (Nv - n) come segue . Forzando n vincoli porta all'eliminazione in una delle n righe corrispondenti ai vertici vincolati (Xi ∈ F ∪ H) e muovendo le colonne corrispondenti al lato destro. L'algoritmo finale di deformazione semplicemente itera due passaggi semplici ed efficaci che sono rispettivamente responsabili per migliorare la stima del locale trasformazioni, e le posizioni di vertice . Ad ogni iterazione , fissato X' , saranno aggiornate le $\hat{\delta}(X_i)$ utilizzando lo step 2 , poi le approssimazioni calcolate saranno utilizzati per risolvere il problema lineare ai minimi quadrati. L'algoritmo per la deformazione non linerare è riportato di seguito(2):

ALGORITHM
Discrete Elastica Nonlinear Deformation (DEND)
INPUT: undeformed vertex set X,
OUTPUT: deformed vertex set X'
Set $X^{(0)} = X$, $\hat{\delta}(X^{(0)}) = L(X)$, k=0
STEP 1 : Compute $\mu^i, \forall X_i \in X$ by (65)
Repeat
    STEP 2 : Compute $\hat{\delta}(X'^{(k+1)})$ by (67) and (68)
    Solve the linear LS problem (64)
    k=k+1
until $\|X'^{(k)} - X'^{(k-1)}\| < 1 \cdot 10^{-3}$

Lo sviluppo della preservazione di dettaglio tramite introduzione di vincoli non lineari è stato approntato ma non è ancora completo e funzionante.

IV.3 OTTIMIZZAZIONE DEL SISTEMA

Il problema di ottimizzazione del sistema si pone solo per la modalità Hard, questo perché i vertici vi ϵ Handled non devono essere parte dell'insieme dei vertici le cui coordinate saranno le incognite del nostro sistema. Contemporaneamente è però necessario che i v_i partecipino al sistema per imporre la loro posizione e la loro topologia rispetto ai propri vicini. Partiamo con definire una A generica di dimensioni $N_v \; x \; N_v$, una X di dimensione N_v e sia preso un B di dimensione N_v (per semplicità supponiamo che X e B siano dei vettori unidimensionali, in realtà essendo formati da coordinate cartesiane saranno di dimensioni $N_v \; x \; 3$).dunque avremo :

$$AX = B \quad \rightarrow \quad \left[a_{ij} \right] * [x_i] \; = [b_i] \tag{69}$$

$$a_{ij} \, \epsilon \, A \qquad i = 1..N_v \; , j = 1..N_v \tag{70}$$

$$x_j \, \epsilon \, X \qquad j = 1..N_v \tag{71}$$

$$b_i \, \epsilon \, B \qquad i = 1..N_v \tag{72}$$

Consideriamo ora :

$$Hndl = \{x_i \mid xi \; coordinate \; di \; vi, \;\; x_i \, \epsilon \, X \; , vi \, \epsilon \, Handled \quad , i = 1..N_v \; \} \tag{73}$$

$$N_H \; = |Hndl| \quad , \;\; N_F \; = \; N_V \; - \; N_H \tag{74}$$

$$iH = \{i \mid x_i \, \epsilon \, Hndl \quad , i = 1..N_H \; \} \tag{75}$$

E definiamo il sistema equivalente :

$$\hat{A} \, \hat{X} = \hat{B} \;\; \rightarrow \;\; \left[\widehat{a_{ij}} \right] * [\widehat{x_i}] \; = \left[\widehat{b_i} \right] \tag{76}$$

$$\hat{A} = \left\{ a_{ij} \;\middle|\; i,j \notin iH \;\; , \;\; a_{ij} \;\; \epsilon A \right\} \tag{77}$$

$$\hat{X} = \left\{ x_j \middle| j \notin iH, \;\; x_j \epsilon X \right\} \tag{78}$$

$$\hat{B} = \{ \, b_i - \textstyle\sum_h a_{ih} x_h \mid \;\; b_i \, \epsilon \, B \; , \;\; a_{ih} \epsilon A \; , \;\; i \notin iH \; , \;\; h \, \epsilon iH \, \} \tag{79}$$

$$\widehat{a_{ij}} \, \epsilon \;\; \hat{A} \quad i = 1..N_F \; , j = 1..N_F \tag{80}$$

$$\widehat{x_j} \, \epsilon \, \hat{X} \quad j = 1..N_F \tag{81}$$

$$\widehat{b_i} \, \epsilon \, \hat{B} \quad i = 1..N_F \tag{82}$$

Notiamo che preso un $\widehat{b_\iota}$, per costruzione sarà :

$$\widehat{b_\iota} = b_i - \sum_h a_{ih}x_h \ , \ ma\ b_i = \sum_h a_{ih}x_h + \sum_f a_{if}x_f \quad con\ h \epsilon\ iH,\ f \notin iH \tag{83}$$

Quindi posso scrivere

$$\widehat{b_\iota} = \sum_h a_{ih}x_h + \sum_f a_{if}x_f - \sum_h a_{ih}x_h = \sum_f a_{if}x_f \tag{84}$$

Che equivale proprio alla formulazione del sistema equivalente. Tuttavia , la matrice $\hat{A}$ non è la matrice di connettività di tale maglia ridotta in quanto i suoi elementi non sono gli stessi che si avrebbero ricalcolando la matrice di connettività originale, sono invece quelli calcolati sull'intera rete.

E' possibile poi introdurre un'ulteriore semplificazione tenendo presente che nel nostro sistema la matrice$A$corrisponde alla matrice topologica laplaciana$L$ o una sua elaborazione che rappresenta interna alla Mesh che di seguito chiameremo E. Notiamo altresì che nel nostro sistema la B non è una costante, ma è generata dal prodotto EX', in cui X' rappresenta la posizione finale di ogni vettore dopo la deformazione, difatti la formulazione del sistema generico era:

$$EX = EX' \ \rightarrow \ nel\ sistema\ presentato\ su: A = E,\ X = X,\ B = EX'$$

Quindi

$$b_i = \sum_j e_{ij}x_j' \quad x_j' \epsilon X',\ e_{ij} \epsilon\ E\ ,\ i = 1..N_v\ , j = 1..N_v \tag{84}$$

Con X'posizione iniziale dei vertici e X posizione finale. Quindi in base alla formulazione del sistema ridotto precedentemente formulato possiamo scrivere

$$\widehat{b_\iota} = b_i - \sum_h e_{ih}x_h = \sum_j e_{ij}x_j{}' - \sum_h e_{ih}x_h = \tag{85}$$

$$\sum_f e_{if}x_f{}' + \sum_h e_{ih}x_h' - \sum_h e_{ih}x_h = \tag{86}$$

$$\sum_f e_{if}x_f{}' + \sum_h e_{ih}(x_h' - x_h) \tag{87}$$

Perciò la nuova formulazione del sistema semplificato sarà:

$$\left[\widehat{e_{ij}}\right] * [\widehat{x_\iota}] = \left[e_{ih} | \ e_{if}\right] * [\ (x_h' - x_h) | \ x_f'\ ] \tag{88}$$

IV.4 INTERFACCIA GRAFICA E INTEGRAZIONE CON BLENDER

Lo sviluppo di un Add-on per Blender presuppone la conoscenza di un minimo di linguaggio di scripting Python e delle strutture fornite da Blender per il supporto allo scripting. Per poter agire efficacemente su strutture matematiche complesse c'è bisogno anche di librerie specializzate che ci permettano di trovare la soluzione più efficace ai problemi che incontriamo. Durante la realizzazione del'applicazione si è dovuto integrare all'ambiente Python il tool di librerie scientifiche scipy. La necessità nasce dall'utilizzo di matrici sparse per contenere la parametrizzazione della superficie M secondo i pesi riportati in sezione IV.1. Per loro natura le matrici sparse ben si prestano a mantenere questo tipo di informazioni, che portano a generare un'enorme quantità di valori 0. Se infatti pensiamo alle colonne della matrice Laplaciana come rappresentative di ognuno dei vertici della mesh ci rendiamo subito conto che il numero di colonne che conterrà sarà N_v(numero di vertici della mesh). Sappiamo però che per costruzione la matrice laplaciana avrà valori diversi da zero solo nelle $|N_{(i)}|$ colonne delle righe relative ai vertici v_i. Dove $N_{(i)}$ rappresenta il numero dei vicini 1-Ring di v_i , dunque $|N_{(i)}|$ è assai limitato. Un altro sforzo continuo nella stesura del codice è quello di mantenere il più possibile il legame con le strutture di Blender per evitare di appesantire la computazione e sfruttare al massimo le potenzialità del sistema. Una delle features che l'ambiente fornisce, ad esempio, è l'indicizzazione univoca di tutti i vertici, di tutte gli edge e di tutte le facce della mesh. Questo permette di poter sempre lavorare sugli indici. Anche i vertex groups di ogni oggetto hanno un indicizzazione univoca, dunque si può definire in maniera semplicissima un dictionary contenente gli indici di ogni vertice appartenente al gruppo. Grazie a questa metodologia è stato possibile definire le regioni Handle, Fixed e Free descritte nella sezione IV.2. Mapperemo le regioni Handle e Fixed su due vertex groups, che potranno essere scelti da interfaccia grafica prima dell'esecuzione della deformazione. La regione Free sarà calcolata come differenza tra i vertici dell'intera mesh e i vertici appartenenti ad Handle e Fixed. Avendo lasciato la scelta delle singole regioni così interattiva non possiamo permetterci di calcolare la matrice Laplaciana per i soli vertici in Free ed Handle. Contestualmente alla creazione della matrice Laplaciana verrà anche creata la matrice diagonale che contiene l'informazione sulla curvatura media. Tutte queste

strutture vanno mantenute anche dopo la deformazione perché rappresentano la mesh nelle sue proprietà geometriche e topologiche. Per gestire il compito di calcolare e memorizzare queste strutture è stato creato un operatore apposito. L'operatore non avrà una sua interfaccia utente e, come descritto ampiamente nella sezione dedicata a Python , può essere richiamato tramite ricerca nella View 3D o tramite pulsante nell'interfaccia utente, in figura 20 (a) è riportato l'activity diagram dell'operatore.

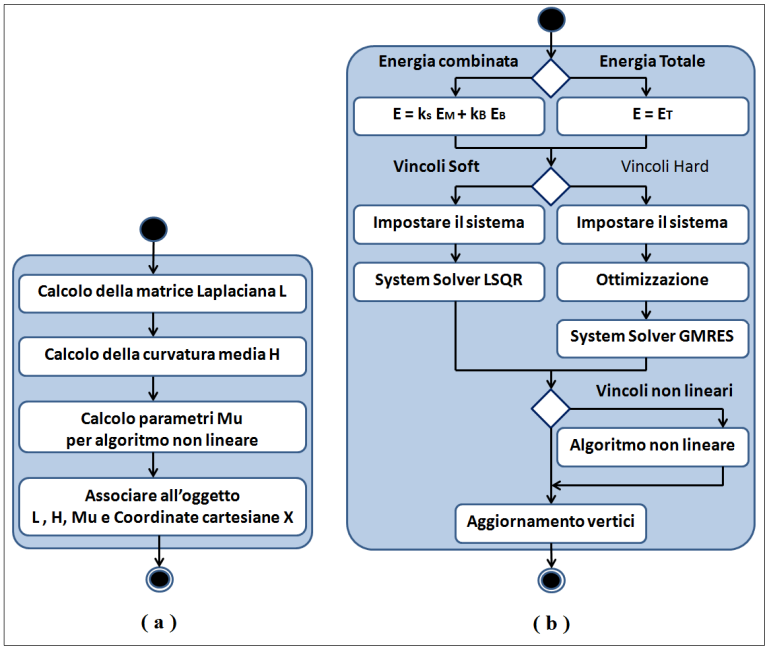

FIGURA 20 ACTIVITY DIAGRAMS DEGLI OPERATORI, (A) FIX SHAPE , (B) DEFORM

Il fatto che l'operatore non abbia una sua interfaccia e possa essere richiamato da diversi punti dell'interfaccia, implica che non ci sia un modo di passargli via codice dei parametri, come farei, ad esempio, se volessi richiamare una semplice funzione. I parametri di cui ha bisogno saranno tutti disponibili all'utente sul Panel creato a cui, ricordiamo, l'operatore non può accedere se sono variabili locali al Panel. La soluzione, che in realtà è la metodologia standard per far interagire l'ambiente con gli operatori, è quella di definire delle proprietà custom sugli oggetti e nel contesto. Le proprietà generate per il nostro add-on sono tutte relative al tipo di deformazione da compiere e al livello di interattività che si è voluto fornire.

Abbiamo dunque introdotto un set di proprietà che vanno a legarsi direttamente ad un qualsiasi oggetto di Blender (vedi sezione III.2.4.6). In questo modo l'operatore andrà ad agire, una volta chiamato, sulle proprietà dell'oggetto che è stato selezionato. Tutto questo può sembrare banale, ma in realtà è molto potente perché ci permette di avere 2 o più oggetti in scena, ognuno dei quali avrà, ad esempio, le sue aree Fixed e Handle, scelte dall'elenco dei suoi vertex groups, o la sua matrice Laplaciana. Quindi possiamo passare da un oggetto all'altro senza perdere le proprietà che sono state scelte o calcolate precedentemente. Questo tipo di implementazione permette all'utente di poter applicare qualsiasi tipo di deformazione sui vertici del gruppo Handle. Per permettere, comunque, di poter definire deformazioni custom è stato realizzato anche un Operatore con interfaccia utente, che applica una deformazione combinando rotazioni e traslazioni sugli assi cartesiani (Figura 21 B). E' stato realizzato infine l'operatore che si occupa di deformare i vertici del gruppo Free, di cui è riportato l'activity diagram in 20(B). L'interfaccia grafica è affidata, invece, ad un Panel che riporta tutte le proprietà necessarie alla deformazione .

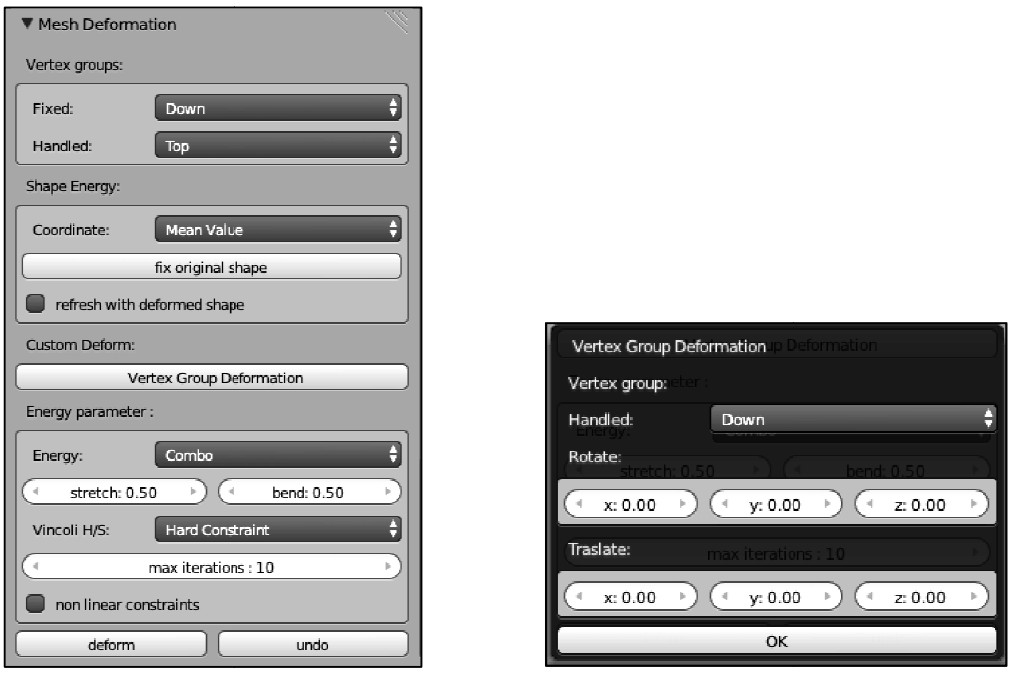

FIGURA 21 (A) GUI ADD-ON **(B)OPERATORE PER DEFORMAZIONI**

Il pannello del nostro Add-on (in figura 21 A) è diviso in zone, disposte dall'alto in basso, logicamente successive nel processo di settaggio di una deformazione. Di seguito una breve descrizione per ogni parte.

IV.4.1.1 VERTEX GROUPS

In questa parte del pannello si possono fissare i vertex group desiderati. Come abbiamo detto ad ognuno dei due insiemi Handle e Fixed può essere associato uno qualsiasi dei vertex groups presenti nell'oggetto selezionato. Il pannello rende disponibile una drop down list in cui appariranno tutti i vertex group del'oggetto. Attenzione, se la lista è vuota significa che l'utente non ha ancora definito alcun vertex group.

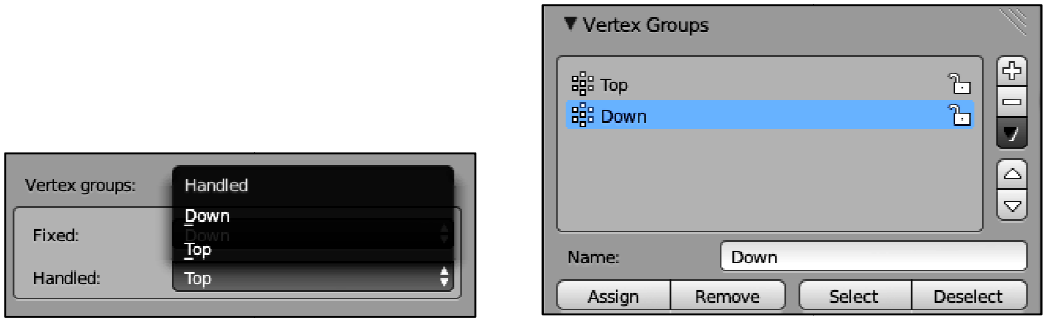

FIGURA 22 – (A) ADD-ON VERTEX GROUPS (B) PANEL BLENDER PER VERTEX GROUPS

La creazione dei vertex group è demandata all'utente, che dovrà semplicemente utilizzare il Panel apposito fornito da Blender (per dettagli su come creare i vertex groups si veda la sezione dedicata ai vertex groups). Nell'esempio in figura 22(B) si può osservare che l'utente ha creato due gruppi etichettati come "Top" e "Down", in 22(A) vediamo le rispettive etichette caricate nella drop down list.

IV.4.1.2 SHAPE ENERGY

Abbiamo bisogno di fissare la forma originale della nostra Mesh per poter avere poi tutti gli elementi necessari alla deformazione.

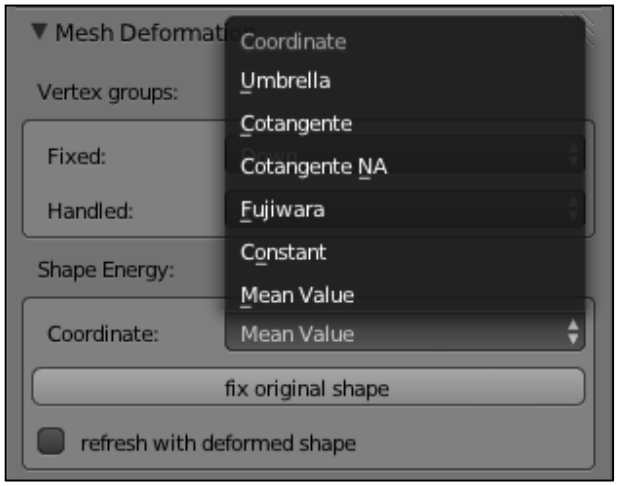

FIGURA 23 FIX SHAPE

Prima di tutto bisogna fissare la mesh mantenendo traccia sia delle sue coordiante cartesiane sia della sua parametrizzazione, attraverso il Laplaciano. Si rende necessario memorizzare anche le coordinate cartesiane perché saranno utilizzate nella risoluzione del Sistema Lineare. Difatti il nostro vettore di spostamento sarà $d = X' - X$ (vedi sezione IV.2) in cui X è il valore delle coordinate cartesiane dei vertici della mesh originale. In figura 23 vediamo la Drop Down List che ci permette di scegliere che tipo di coordinate parametriche usare per il calcolo di L. Oltre alla forma fissiamo anche la curvatura media G e manteniamo anche traccia dei dettagli geometrici calcolando i µi derivati dalle normali sulle facce della mesh (vedi sezione Vincoli Non Lineari). Si da infine anche la possibilità di aggiornare tutte le informazioni geometriche e topologiche della mesh al termine dell'esecuzione di ogni deformazione .

IV.4.1.3 ENERGY PARAMETER

Restano da definire la tipologia della deformazione e tutte le proprietà che si rendono necessarie per calcolarla. Il Panel cambia assetto rispetto alla tipologia di deformazione scelta. In figura 24, sulla destra si può osservare come, per l'utilizzo dell'energia che nasce dalla combinazione tra energia di stretching ed energia di bending, è possibile settare le variabili stretch e bend , che altro non sono che k_b k_s descritte nel modello risolutivo.

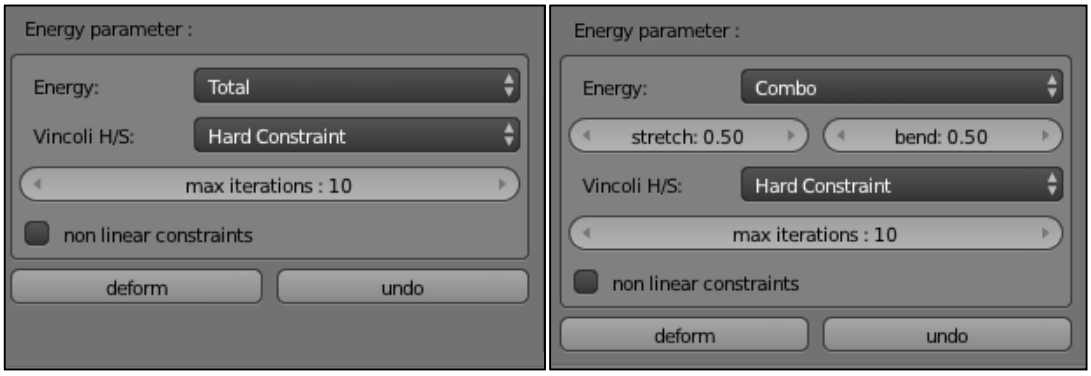

FIGURA 24 - ENERGY PARAMETER

Un ulteriore elemento di scelta è la gestione dei vincoli. Ricordiamo che la scelta di un vincolo Soft implica la creazione di un sistema sovradimensionato e dunque l'operatore che gestisce la deformazione dovrà utilizzare il risolutore LSQR(Least Squared) (Sezione IV.2) come è possibile vedere nel suo diagramma delle attività (Figure . Nel caso della scelta di vincoli Hard abbiamo, nel sistema risolutivo, una matrice A quadrata e l'operatore risolverà il sistema con l'algoritmo GMRES (Generalized Minimal Residual Method). Il pannello fornisce già il supporto all'introduzione di vincoli non lineari. Il pulante deform è legato all'operatore che calcolerà X' , cioè le coordinate cartesiane dei vertici a deformazione avvenuta. Una volta calcolate si occuperà anche di applicarle ai vertici della Mesh. L'operatore legato al pulsante undo ripristina, invece, la posizione originale.

V. RISULTATI SPERIMENTALI

L'Add-on risponde efficacemente alla gestione di deformazioni libere e permette all'utente la libertà completa in questo campo. Il supporto all'utilizzo di una struttura così ben radicata in Blender come i vertex groups, che ci permettono di mantenere selezioni, applicare modificatori e materiali, direttamente su un sottoinsieme della Mesh, è stata di certo una delle vittorie di questo percorso. Come ci si aspettava, lavorando su un linguaggio interpretato, i tempi non sono quelli che ci si possono attendere da un codice nativo, d'altro canto la flessibilità di Python rende l'estensione molto fruibile, immediata e per questo risulta un supporto ottimo per l'ambiente accademico. I tempi qui presentati sono stati presi su una macchina con procesore Intel®Core™2- T7200 a 2GHz.

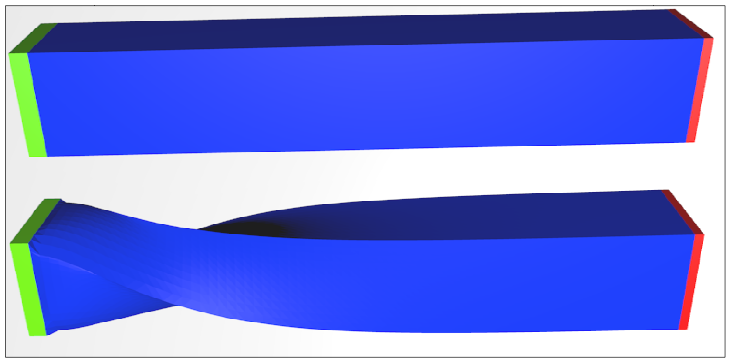

FIGURA 25 ROTAZIONE DI 90 GRADI SU MESH BAR

Mesh bar (3762 vertexs) - Handle: 210 - Fixed: 96 - Free: 3456								
Constraint	**Deformation**	**System Solver**	**Tolerance A**	**Tolerance B**	**Max iterations**	**Iterations**	**Time per coord (sec)**	**Time Total (sec)**
hard	rot 30°	gmres	1,00E-05		50	<50	6,189	**20,085**
hard	rot 90°	gmres	1,00E-05		50	<50	6,383	**20,920**
soft	rot 90°	lsqr	1,00E-04	1,00E-04	n.a.	449,2	7,016	**23,090**
soft	rot 90°	lsqr	1,00E-05	1,00E-05	n.a.	2696,7	55,370	**169,010**
soft	rot 90°	lsqr	1,00E-06	1,00E-06	n.a.	6739,333	132,033	**399,550**
soft	rot 90°	lsqr	1,00E-06	1,00E-03	n.a.	6051	135,466	**435,880**

TABELLA 1 - TIMING PER DEFORMAZIONI SU MESH BAR

I tempi aumentano proporzionalmente alla dimensione del sottoinsieme della mesh, che definisce le variabili del nostro sistema risolutivo, cioè l'insieme Free (sezione IV.2). Per elaborazioni molto complesse, infatti, i tempi tendono a lievitare, come già accade per modificatori di tipo nativo in Blender. Un esempio è la precomputzione delle coordinate

armoniche nel Mesh Deform Modifier (sezione III.1.2.3). Dettagliatamente, rispetto alla deformazioni implementate e applicabili, le tempistiche sono favorevoli ad un approccio ai vincoli di tipo hard-constraint rispetto al soft. La motivazione è di certo da ricercare nell'algoritmo di risoluzione che può avvantaggiarsi del fatto di avere una matrice delle energie di dimensioni più ridotte. La deformazione definita con vincoli soft, infatti, nasce già con un sistema di risoluzione sovradimensionato che deve essere risolto con i minimi quadrati(sezione IV.2). Il più delle volte, anche definire una tolleranza molto bassa, come ad esempio 1e-06, non basta a far trovare una soluzione ottimale all'algoritmo least squared (IV.4.1.3), che risolve il sistema definito per vincoli di tipo soft. Chiaramente quanto più piccola sarà la tolleranza, tanto maggiore risulterà il tempo di computazione e le iterazioni che l'algoritmo dovrà compiere, come è chiaramente osservabile in Tabella 1. In questo caso, ad esempio, sono stati elencati anche i tempi riscontrati relativamente a differenti tolleranze. L'algoritmo GMRES (IV.4.1.3), che risolve il sistema nel caso di vincoli di tipo Hard risulta molto più performante, seppur con tolleranza di default a 1e-05. Già con un numero di iterazioni vicino a 50 riesce a risolvere quasi tutte le deformazioni fornitegli. La tabella 1 riporta i tempi relativi ad una rotazione applicata alla mesh `bar` (3762 vertici). In figura 25 possiamo osservare una rotazione applicata ruotando il vertex group relativo ai vertici handle (verde) e tenendo fisso il vertex group relativo ai vertici fixed (rosso). Useremo tale convenzione di colori anche per le deformazioni presentate in figura 26 e 27. In tabella 2 riportiamo i tempi relativi all'applicazione di una deformazione definita da user interface sulla mesh `dino`, in particolare per muovere la coda del nostro personaggio, come si può osservare in figura 26, l'insieme Fixed coprirà la quasi totalità del personaggio lasciando libera solo la coda, difatti comprenderà 8034 vertici su 10098.

Mesh dino (10098 vertexs) - Handle: 430 - Fixed: 8034 - Free: 1634								
Constraint	**Deformation**	**System Solver**	**Tolerance A**	**Tolerance B**	**Max iterations**	**Iterations**	**Time per coord (sec)**	**Time Total (sec)**
hard	libera	gmres	1,00E-05		50	50	9,867	**40,620**
hard	libera	gmres	1,00E-05		100	100	27,295	**47,629**
soft	libera	lsqr	1,00E-05	1,00E-05	n.a.	951,77	11,877	**44,546**
soft	libera	lsqr	1,00E-06	1,00E-06	n.a.	2049	29,008	**96,776**

TABELLA 2 TIMING PER DEFORMAZIONI SU MESH DINO

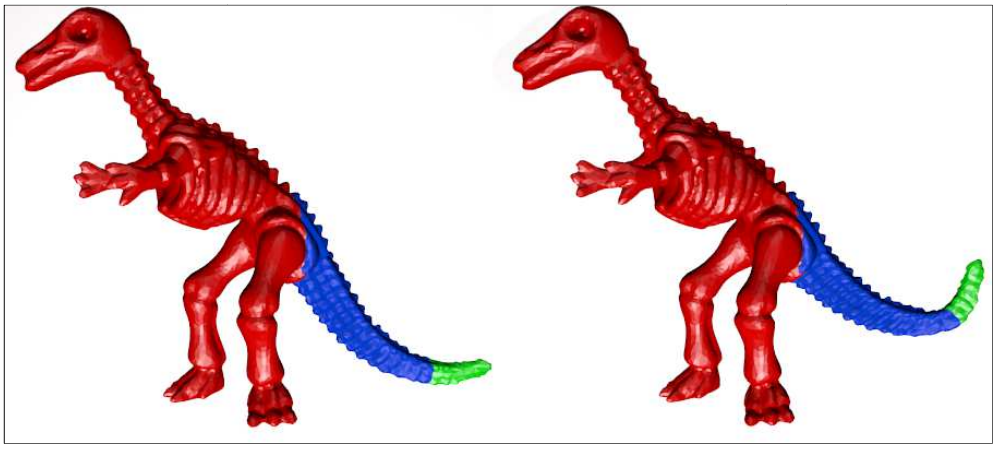

FIGURA 26 DEFORMAZIONE LIBERA SULLA MESH DINO

Quanto detto è ampiamente dimostrato dalla tabella 3 che riporta i tempi relativi a una semplice rotazione dei vertici appartenenti al'insieme Handle (sezione IV.2) della mesh bumped plane (Figura 27) . L'algoritmo GMRES relativo al caso dei vincoli hard, arriva alla soluzione completa con meno di cinquanta iterazioni, mentre il sistema definito per i vincoli soft viene risolto con almeno 2025 iterazioni. C'è da dire comunque che il livello di tolleranza da applicare è molto dipendente dalla morfologia della mesh da deformare e non solo dal numero dei suoi vertici.

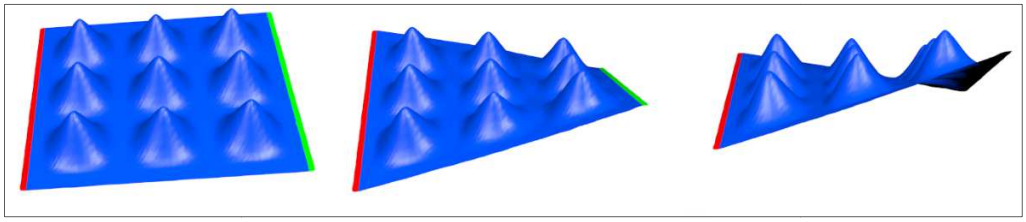

FIGURA 27 ROTAZIONE DI 30° SU MESH BUMPED PLANE

Mesh bumped plane (2115 vertexs) - Handle: 180 - Fixed: 90 - Free: 1845								
Constraint	**Deformation**	**System Solver**	**Tolerance A**	**Tolerance B**	**Max iterations**	**Iterations**	**Time per coord (sec)**	**Time Total (sec)**
hard	rot30°	gmres	1,00E-05		50	<50	2,276	**7,690**
soft	rot30°	lsqr	1,00E-06	1,00E-06	n.a.	2025	20,297	**62,659**
soft	rot30°	lsqr	1,00E-07	1,00E-07	n.a.	2025	22,699	**69,809**

TABELLA 3 TIMING PER DEFORMAZIONI SU MESH BUMPED PLANE

Appurato questo, dobbiamo dire che la deformazione tramite elasticità discreta ha dato ottimi risultati se confrontati con altre tipologie di deformazione.

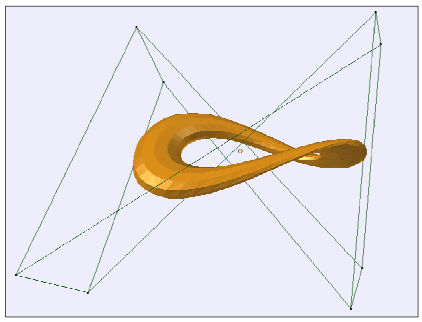

FIGURA 28 DEFORMAZIONE CON MODIFICATORE LATTICE

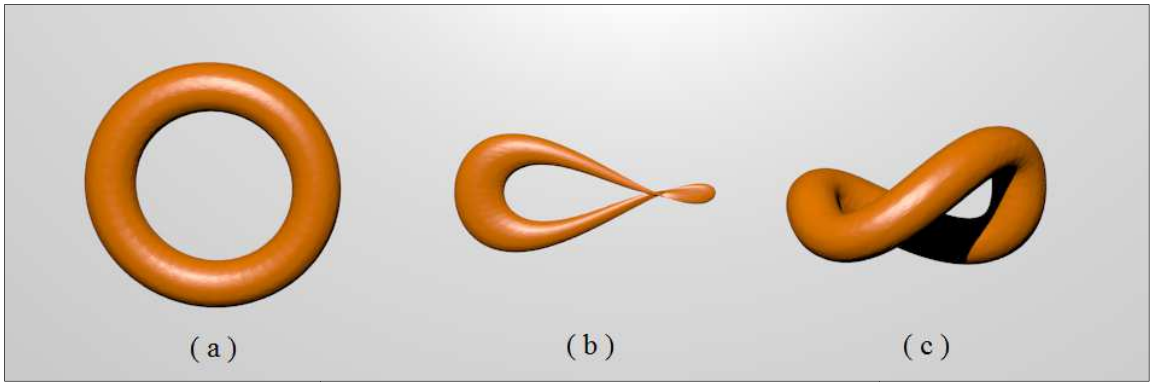

FIGURA 29 TORSIONE CON MODIFICATORE LATTICE E CON ELASTICITÀ DISCRETA

Il figura 29 è riportato un esempio di torsione sulla mesh `torus` (576 vertici). Si prenda a riferimento la prima immagine (a) che riporta la mesh al suo stato originale. Per ottenere la prima deformazione (b) è stato applicato un modificatore di tipo Lattice (III.1.2.2), immergendo la mesh in un reticolo con 8 punti, osservabile in figura 28, a cui è stata applicata una torsione. Si può notare come il volume della mesh sia stato modificato a seguito della deformazione pur avendo impostato il suo mantenimento nelle property del modificatore. Nella terza immagine (c) è stata applicata la torsione tramite elasticità discreta. Risulta subito visibile come questo tipo di deformazione riesca ad applicare la deformazione mantenendo inalterate le proprietà topologiche della mesh.

Nell'esempio in figura 30 è riportato il risultato di una rotazione applicata sul braccio sinistro del personaggio rappresentato dalla mesh `dino`, per portarci ad una posa in cui il nostro personaggio risulterà con la mano rivolta verso l'alto. Nel caso (c) abbiamo creato uno scheletro parziale che ci permetterebbe di muovere il braccio sinistro per intero, la

spalla sinistra ed il tronco del personaggio. Applicando la rotazione del braccio possiamo notare come la deformazione di questo tipo vada a rovinare la struttura geometrica della spalla del personaggio.

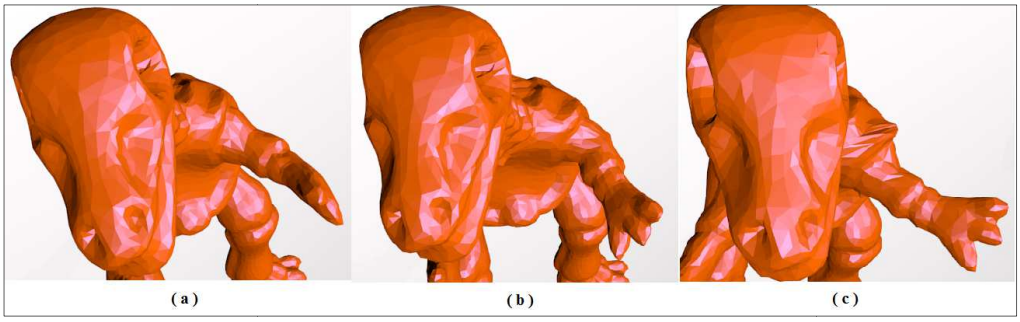

FIGURA 30 ROTAZIONE MANO CON ELASTICITÀ DISCRETA (B) E CON ARMATURE MODIFIER(B)

Nel caso di una deformazione di tipo surface based (b) si può osservare invece che, pur avendo ruotato il braccio del personaggio, la posa risultante pare non abbia influito negativamente sulla sua spalla e ne abbia mantenuto l'aspetto originario.

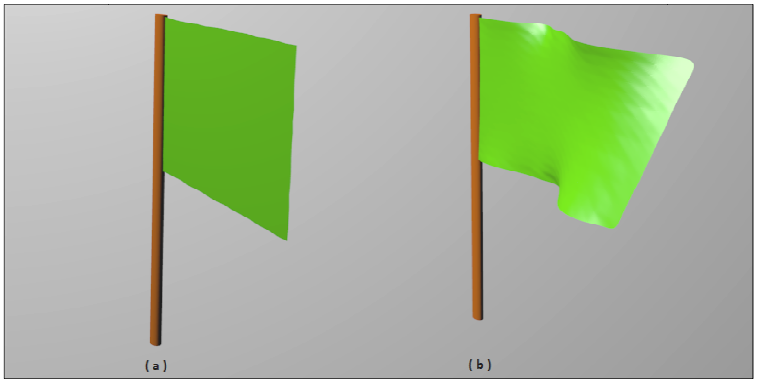

FIGURA 31 DEFORMAZIONE DI MESH FLAG

In figura 31 è riportato un esempio di deformazione su mesh flag (289 vertici) applicata fissando i vertici sul lato sinistro e muovendo il lato destro.

Infine, in figura 32 possiamo vedere il risultato dell'applicazione di una deformazione sulla mesh `dino` (10098 vertici), la posa è stata definita interattivamente posizionando il vertex group di tipo handle nella posa desiderata, fissando la parte che non doveva muoversi con un ulteriore vertex group ed infine applicando l'algoritmo di deformazione. Si può notare come le pose finali mantengano una perfetta naturalezza.

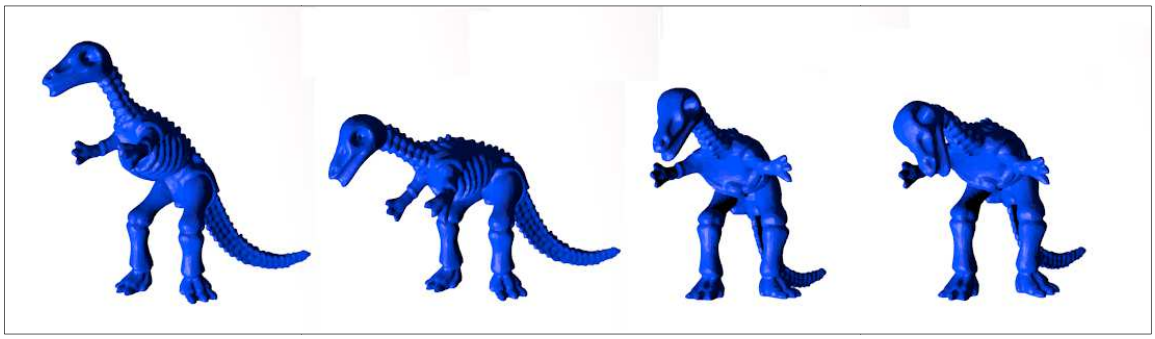

FIGURA 32 DEFORMAZIONI APPLICATE INTERATTIVAMENTE SULLA MESH "DINO"

Concludiamo andando a richiamare l'attenzione sul fatto che le tolleranze applicate ai singoli algoritmi risultano particolarmente importanti rispetto al tipo di mesh da deformare. Come possibile estensione dell'Add-on proponiamo dunque di aggiungere questa Property all'insieme già definito, per permettere un uso più accurato rispetto all'oggetto a cui si applica la deformazione. Un ulteriore passo sarebbe quello di completare la realizzazione dei vincoli non lineari (IV.2.1.1) che mantengono i dettagli geometrici sulla mesh. Per la definizione data in sezione IV.2.1.1, il sistema risolutivo che li implementa dovrà prevedere la risoluzione di un sistema sovradimensionato, quindi l'uso del'algoritmo LSQR. Si consiglia di trovare un modo di ottimizzare tale sistema come già e successo nel caso dei vincoli hard, per evitare di appesantire ulteriormente i tempi di deformazione.

VI. BIBLIOGRAFIA

1. **Botsh M., Kobbelt L. , Pauly M. , Alliez P., Lévy B.** *Polygon Mesh Processing.* s.l. : A.K.Peter ,Ltd, 2010.
2. *Deformation by discrete elastica.* **F.Bertini, S.Morigi.** s.l. : EUROGRAPHIC ASSOCIATION, 2013, pp. 223 - 232.
3. **T.J., Wilmore.** *Total Cuvature in Riemannian Geometry.* New Jork : John Wiley and Sons, 1982.
4. **M.P., do Carmo.** *Riemannian Geometry.* Boston-Basel-Berlin : Birkhauser, 1993.
5. *Linear Varational Surface Dedformation Methods.* **O., Botsch M. and Olga Sorkine.** 2008, IEEE Trans. on Visualization and Computer Graphics,14/1, pp. 213-230.
6. *Mesh Forging: Editing of 3D-Meshes Using Implicity Defined Occluders.* **R.Klein, g.H.Bendels and.** Aire-la-Ville, Switzerland:Eurographics Association : s.n., 2003. Proc. of Eurographics Symposium on Geometry Processing. pp. 207-217.
7. **Pauly, Mark.** Point Primitives for Interactive Modelling and Processing of 3D Geometry. *PhD Thesis.* ETH Zurich,Konstanz,Germany:Hartung Gorre : s.n., 2003.
8. *Elastically deformable models.* **D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer.** New York : ACM : s.n., 1987, Computer Graphics, 21(4), pp. 205-14.
9. *Deformable Curve and Surface Finite-Elements for Free-Form Shape Design.* **Gossard, G. Celniker and D.** New York: ACM : s.n., 1991. Proc. of ACM SIGGRAPH. pp. 257-66.
10. *Free-Form Deformation of Solid Geometric Models.* **S. R. Parry, T. W. Sederberg.** New York: ACM : s.n., 1986. Proc. of ACM SIGGRAPH. pp. 151-159.
11. *Scattered Data Approximation.* **H.Wendland.** Cambridge,UK : s.n., 2005, Cambridge University Press.
12. **Blender.** QuickStart Introduction , Blender v2.59.0 –API documentation. [Online] http://www.blender.org/documentation/blender_python_api_2_59_0/info_quickstart.html.

13. **Wiki Blender.** Shaping a Fork (Lattice Modifier),Doc:2.6/Tutorial. [Online] http://wiki.blender.org/index.php/Doc:2.6/Tutorials/Modifiers/Lattice/Shaping_a_Fork.
14. **Blender.** Blender documentation contest. [Online] http://www.blender.org/documentation/blender_python_api_2_67_1/.
15. *Harmonic Coordinates for Character Articulation, Pixar Technical Memo #06-02b,.* **Pushkar Joshi Mark Meyer, Tony DeRose, Brian Green Tom Sanocki.** s.l. : Pixar Animation Studios., 2007.
16. **Anders, Michel.** Creating a Blender 2.6 Python add-on. [Online] http://michelanders.blogspot.it/p/creating-blender-26-python-add-on.html.
17. **Sorkine, Olga.** *Laplacian Mesh Processing.* s.l. : Eurographics Association, 2005.
18. **Wiki Blender .** Script in python, Manual Blender. [Online] http://wiki.blender.org/index.php/Doc:IT/2.4/Manual/Extensions/Python.
19. **Wiki Blender.** Mesh Deform Modifier, Doc:2.4/Manual/Modifier/Deform. [Online] http://wiki.blender.org/index.php/Doc:2.4/Manual/Modifiers/Deform/MeshDeform.
20. —. Mesh Deform Modifier, Doc:2.6/Manual/Modifier/. [Online] http://wiki.blender.org/index.php/Doc:2.6/Manual/Modifiers/Deform/Mesh_Deform.
21. **Meyer M., Desbrun M .,Schroder P., Barr A. H .** *Discrete differential-geometry operators for triangulated 2-manifolds.* 2003. Vol. Visualization and Mathematics III.
22. *Mean value coordinates for closed triangular meshes.* **Ju T., Schaefer S.,Warren J.** 2005, Vols. ACM Trans.Graph. 24, 3, pp. 561–566.

-