Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea Triennale in Informatica

IADO:
un pacchetto per l'animazione e deformazione di curve e superfici

Relatore:
Chiar.mo Prof. Giulio Casciola

Presentata da:
Stefano Montefiori

Terza Sessione
Anno Accademico 2010-2011

Indice

Elenco delle figure

Introduzione

Gli oggetti *NURBS*, seppur affermati a livello industriale per la rappresentazione e il design, svariando dalla progettazione cad ai videogame, rappresentano solo un *modello geometrico*; una interessante generalizzazione verso un modello fisico sono le *DNURBS* (Dynamic Nurbs), che incorporano al loro interno i principali parametri fisici, quali densità di massa, densità di dissipazione dell'energia, energie di deformazione interna ed alcuni altri, permettendo quindi di simulare l'evoluzione della geometria in presenza di particolari condizioni.

Il modello, pur essendo molto potente, ha come obiettivo quello di fornire una simulazione di eventi pseudo-realistica e non sempre quindi ci si deve aspettare un comportamento identico a quello osservabile nel mondo reale.

Il modello delle *DNURBS a pesi costanti nel tempo* è quello utilizzato dal pacchetto dncurv/dnsurf, il vero e proprio motore di calcolo del sistema, che simula l'evoluzione di una curva/superficie nel tempo secondo i parametri fisici impostati, al quale IADO fa da front-end fornendo all'utente una interfaccia interattiva con la quale operare in modo semplice e intuitivo.

Nel primo capitolo vengono fornite alcune nozioni teoriche necessarie per comprendere le basi matematiche su cui si basa il modello fisico delle DNURBS implementato da *dncurv/dnsurf*, e capire meglio come i suoi parametri agiscono nell'evoluzione temporale che porta alla deformazione della geometria.

Nel secondo capitolo sarà discussa la struttura di IADO in ogni sua funzionalità, cercando di fornire al prossimo utilizzatore tutti gli elementi necessari per muoversi con consapevolezza nell'interfaccia stessa, anticipando le

modifiche apportate in questa ultima versione.

Essendo il mantenimento di entrambi i pacchetti, *Iado* e *dncurv/dnsurf*, fermo da circa nove anni, i sorgenti non erano in grado di produrre alcun eseguibile con i compilatori oggi in uso. Visto che stiamo assistendo nuovo picco di interesse verso il mondo delle DNURBS da parte della Computer Graphics in generale, la fase operativa di questo lavoro, e quindi il suo terzo capitolo consiste proprio nel rendere di nuovo utilizzabili *dncurv/dnsurf* e *Iado* nelle loro complete funzionalità, allineando il codice sorgente ai nuovi standard che il compilatore richiede e modernizzando le librerie esterne a cui alcune parti di codice si appoggiano. In un secondo momento verranno analizzati i punti deboli del codice e della interfaccia, operando modifiche atte a migliorare la stabilità di utilizzo e la semplicità d'uso.

Il quarto capitolo comprende una analisi sperimentale: effettuando diverse simulazioni si cerca di mettere in evidenza sperimentalmente come agiscono i vari parametri e come la deformazione della geometria evolve alla loro modifica.

Infine, nelle considerazioni verrano discussi lati positivi e limiti di questo approccio, e si fornirà spunto per eventuali sviluppi futuri.

Capitolo 1

Curve e superfici NURBS e DNURBS

In questo primo capitolo ci soffermeremo a discutere del modello matematico che sta alla base della rappresentazione grafica di curve e superfici, cioè le *NURBS*, passando poi al modello fisico utilizzato dal motore di calcolo del pacchetto: le *Dynamic NURBS*, o *DNURBS*.

1.1 NURBS

NURBS è l'abbreviazione *Non Uniform Rational B-Splines*, traducibile in "B-Splines razionali non uniformi"; rappresentano una classe di curve geometriche utilizzate in computer grafica per rappresentare curve e superfici.

Una NURBS è quindi la rappresentazione matematica che un software crea di un oggetto, per definirne accuratamente la forma.

Rappresentano il substrato geometrico delle Dynamic NURBS, con le quali condividono la quasi totalità della struttura, fatta eccezioni per quei parametri che generalizzano le DNURBS verso un modello fisico.

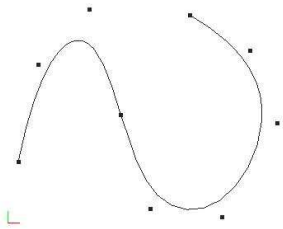

Figura 1.1: Esempio di curva NURBS

1.1.1 Forma Implicita e Parametrica

I due principali modelli matematici per la rappresentazioni di curve e superfici sono:

- *Forma Implicita*

  L'equazione di una curva che giace sul piano xy ha la forma:

  $$f(x, y) = 0 \tag{1.1}$$

  L'equazione è unica e implica una relazione fra le coordinate x e y dei punti della curva

- *Forma Parametrica*

  Ognuna delle coordinate di un punto della curva è rappresentato separatamente come funzione esplicita di un parametro indipendente:

  $$C(u) = (x(u), y(u)) \;\; a \leq u \leq b \tag{1.2}$$

Ognuna delle due forme ha pro e contro che vanno presi in considerazione a seconda dell'ambito di applicazione. In Computer Graphics la rappresentazione parametrica ha riscosso molto più successo, per i seguenti motivi:

- Le curve rappresentate in forma parametrica, a differenza di quelle in forma implicita, possiedono una naturale direzione di attraversamento che facilita di molto la generazione di sequenze ordinate di punti lungo la curva.

- L'*intervallo di definizione* fa parte della definizione stessa del modello parametrico, ed in questa forma è quindi facile rappresentare segmenti di curva limitati. E' tuttavia altrettanto semplice, in forma implicita, rappresentare geometrie non limitate, vantaggio inutile nel nostro ambito.

- La forma parametrica risulta più naturale per rappresentare geometrie in Computer Graphics, a causa dell'intuitivo significato geometrico che i coefficienti di molte funzioni parametriche hanno. Questo porta anche alla creazione di algoritmi numericamente più stabili.

- Il modello parametrico è facilmente estendibile alle tre dimensioni dello spazio, aggiungendo il parametro z

- Calcolare un punto della curva (o della superficie) in forma implicita è molto complesso rispetto alla forma parametrica. In forma implicita però è nettamente più semplice, rispetto alla rivale, determinare se un determinato punto appartiene alla curva/superficie, ma non è un vantaggio sfruttabile nel nostro ambito.

1.1.2 Polinomi e B-Splines

Per la parametrizzazione, vi sono alcuni aspetti di cui occorre tener particolarmente conto:

- il modello usato deve poter permettere all'utente di rappresentare tutte le curve o superfici desiderate

- l'errore in fase di calcolo deve essere il più piccolo possibile

- vi sia un metodo semplice per la loro memorizzazione e per il loro calcolo

Il metodo più semplice è quello dei polinomi: sono molto efficienti dal punto di vista computazionale e accurati nell'approssimazione numerica nel caso di curve di Bezier. Soffrono però anche di alcune problematiche: una modellazione interattiva della geometria è molto difficile, una modifica non è locale ma influenza tutta la curva, è necessario un polinomio di grado $n+1$ per interpolare n punti. Questi problemi sono principalmente dovuti all'utilizzo di un solo polinomio per la rappresentazione dell'intera curva, e vengono in parte risolti dalle curve polinomiali a tratti, ossia dall'uso di un polinomio indipendente per ogni tratto di curva.

Un altra classe di funzioni efficienti in questo ambito sono le *B-Spline*, di cui viene data la definizione ricorsiva:

$$u(x) = \begin{cases} 1 & \text{se } u_i \leq u < u_{i+1} \\ 0 & \text{altrimenti} \end{cases} \tag{1.3a}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{1.3b}$$

Curve e superfici descritte da B-Spline mantengono tutti i pro delle curve polinomiali, ponendo però una pezza ai suddetti problemi, in particolare alla modifica locale, che coinvolge solo il tratto $[u_i, u_{i+p+1})$, poichè $N_{i,p} = 0$ per $u \notin [u_i, u_{i+p+1})$

1.1.3 Non Uniform Rational B-Splines

Rimangono alcune curve e superfici la cui rappresentazione non è direttamente modellabile con B-Spline o polinomi: fra queste abbiamo ad esempio le coniche come cerchi, parabole o sfere. Da questo problema nascono appunto le *Non Uniform Rational B-Splines*, abbreviate con la sigla NURBS. Prima di darne una definizione chiara, occorre definire le *funzioni razionali di base*:

$$R_{i,p}(u) = \frac{N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j} \tag{1.4}$$

Una curva NURBS può essere quindi vista come:

$$C(u) = \sum_{i=0}^{n} R_{i,p}(u) P_i \tag{1.5}$$

dove:

- P sono i controlpoints e formano il poligono di controllo
- w_i sono i pesi associati
- $N_{i,p}(u)$ sono le funzioni B-Splines di base definite sul vettore periodico e non uniforme dei knots:

$$U = \{\underbrace{a, ..., a}_{p+1}, u_{p+1}, ..., u_{m-p-1}, \underbrace{b, ..., b}_{p+1}\} \tag{1.6}$$

  con $a = 0$, $b = 1$, e $w_i > 0$ per ogni i

Analogamente a quanto accade per le curve, una superficie NURBS di grado p nella direzione u e grado q nella direzione v è definita come:

$$S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u, v) P_{i,j} \tag{1.7}$$

dove:

- $\{ P_{i,j} \}$ sono i controlpoints che formano la "griglia di controllo bidirezionale"
- $\{ w_{i,j} \}$ sono i pesi associati
- $R_{i,j}(u, v)$ sono le funzioni razionali di base definite come:

$$R_{i,j}(u, v) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n} \sum_{l=0}^{m} N_{k,p}(u) N_{l,q}(v) w_{k,l}} \tag{1.8}$$

  Anche in questo caso gli $\{ N_{i,p}(u) \}$ e gli $\{ N_{j,q}(v) \}$ sono le B-Splines definite sui vettori di knot:

$$U = \{\underbrace{0, ..., 0}_{p+1}, u_{p+1}, ..., u_{r-p-1}, \underbrace{1, ..., 1}_{p+1}\} \tag{1.9a}$$

$$V = \{\underbrace{0, ..., 0}_{p+1}, v_{q+1}, ..., u_{m-p-1}, \underbrace{1, ..., 1}_{p+1}\} \tag{1.9b}$$

con $r = n + p + 1$ e $s = m + q + 1$.

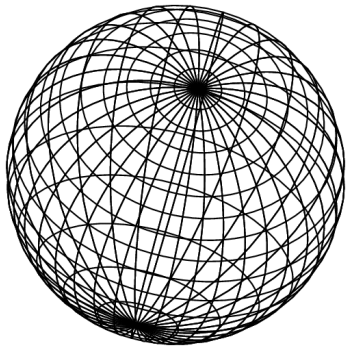

Figura 1.2: Esempio di sfera NURBS

1.2 DNURBS

Il limite delle NURBS risiede nel fatto che, pur permettendo di rappresentare una vastissima gamma di curve e superfici, richiedono all'utente di modificare manualmente pesi, posizione dei punti di controllo e gradi di libertà per ottenere la forma voluta, processo poco intuitivo e laborioso.

Nel 1994 Terzopoulos e Qin, con il lavoro in[TEQI94], propongono le Dynamic Nurbs come strumento per realizzare una modellazione dinamica. Esse infatti rappresentano un modello fisico che incorpora i parametri formulati dalla meccanica Lagrangiana, quali distribuzione di massa, energie di deformazione interna, forze ed altri valori. L'utente, grazie a questo strumento, può ora modellare superfici e curve applicando forze e vincoli alla geometria, che,

mediante la base numerica formata dalle equazioni differenziali, produrranno risulti fisici significativi, deformando la nurbs.

Grazie all'introduzione delle DNURBS a pesi constanti nel tempo, introdotta in [CARU98], abbiamo un ulteriore miglioramento che porta il modello ad adattarsi alla simulazione realistica di fenomeni fisici.

1.2.1 Curve e superfici DNURBS

Una DNURBS, definita come nella (1.5), è vista in funzione del tempo t come:

$$C(u,t) = \sum_{i=0}^{n} P_i(t) R_{i,p}(u,t) \tag{1.10}$$

con

$$R_{i,p}(u,t) = \frac{N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j(t)} \tag{1.11}$$

Analogamente definiamo le superfici DNURBS, sempre parametrizzate nel tempo t:

$$S(u,v,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v,t) P_{i,j}(t) \tag{1.12}$$

con

$$R_{i,j}(u,v,t) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}(t)}{\sum_{k=0}^{n} \sum_{l=0}^{m} N_{k,p}(u) N_{l,q}(v) w_{k,l}} \tag{1.13}$$

1.2.2 Equazioni del moto

Il modello fisico delle DNURBS si basa sulla versione lavoro-energia della dinamica Lagrangiana. Viene preso in considerazione un sistema fisico astratto: sia $p(t)$ un insieme di coordinate generalizzate. Se viene applicata una forza $f_i(t)$ ai punti di controllo P_i, possono essere ricavate le equazioni del moto Lagrangiano:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{p}_i} - \frac{\partial T}{\partial p_i} + \frac{\partial F}{\partial \dot{p}_i} + \frac{\partial U}{\partial p_i} = f_i \tag{1.14}$$

dove T è l'energia cinetica, u è l'energia potenziale ed F è l'energia di dissipazione di Raleigh. Usando quindi la definizione appena vista è possibile formulare un modello fisico, partendo da una NURBS e introducendo i parametri descritti. Senza entrare troppo nel complesso modello matematico, è possibile ricavare le matrici:

- M di inerzia
- D di dissipazione
- K di rigidità

che combinate definiscono le equazioni di moto. Sempre in [TEQI94] vengono definiti forze e vincoli meccanici, molto importanti ai fini della modellazione: è infatti possibile deformare una geometria applicandovi una forza oppure assicurandone una parte a dei vincoli e definendo un certo coefficiente elastico alla struttura.

1.2.3 DNURBS a pesi costanti nel tempo

Il modello proposto in [TEQI94] presenta però purtroppo alcuni limiti a livello fisico/geometrico (non rendendolo particolarmente adatto ad una simulazione pseudo-realistica) e a livello computazionale (compromettendone l'implementazione). Le principali critiche vengono ampiamente discusse in [CARU98], dove viene anche proposta una soluzione che è poi quella adottata in *dncurv/dnsurf*: le *DNURBS a pesi costanti nel tempo*. Essa propone di variare liberamente i controlpoints, ma mantenere i pesi ad essi associati costanti nel tempo:

$$c(u,t) = \sum_{i=0}^{n} P_i(t) R_{i,p}(u) \tag{1.15}$$

dove $R_{i,p}(u)$ è data in (1.4).

Mentre una superficie è definita come:

$$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v) P_{i,j}(t) \tag{1.16}$$

con $R_{i,j}(u,v)$ definito in nella (1.5).

Con questa implementazione, le suddette matrici M, D e K sono calcolate analogamente ma restano costanti nel tempo, sia per quanto concerne le curve che le superfici. Viene inoltre introdotto il concetto di *vincolo geometrico*, che risolve diversi inconvenienti presenti nella versione originale, in particolare:

- i pesi associati ai controlpoints non sono più coinvolti come parametro Lagrangiano, eliminandone i problemi
- occorre calcolare una sola volta le matrici M e D, in quanto risultano inferiori, meglio condizionate e rimangono costanti.

Queste soluzioni rendono le DNURBS a pesi constanti nel tempo efficienti ed utili per effettuare deformazioni pseudo-realistiche di curve e superfici, permettendo di modellarle e animarle interattivamente con un costo computazionale non esagerato.

1.3 Pacchetto *dncurv/dnsurf*

In [RUBI96], sulla base di quanto detto sulle DNURBS a pesi costanti nel tempo, è stato implementato il pacchetto *dncurv/dnsurf* che accetta come argomenti alcuni parametri e, sulla base di essi, crea una evoluzione della geometria di partenza, generando un numero arbitrario di nuove curve/superfici. Iado si appoggia interamente a questi due eseguibili per il calcolo dei passi di simulazione. I parametri fisici che partecipano alla deformazione della nostra curva/superficie sono:

- α: Controllando le tensioni locali, questi valori, se diversi da 0, caricano di energia potenziale elastica la nostra geometria, che durante la fase di simulazione può trasformarsi in energia cinetica, avvicinando tra loro i

controlpoints come se avessimo a che fare con un materiale elastico. Per le superfici è possibile suddividere il parametro in u e v, cioè specificare valori diversi di $\alpha_{1,1}$ e $\alpha_{2,2}$.

- β: Hanno il significato opposto rispetto agli α, controllando la rigidità dell'oggetto: più è alto il loro valore, più la geometria, sottoposta a una forza esterna o elastica, tenderà a rimanere rigida. E' possibile suddividere il parametro in u ($\beta_{1,1}$), v ($\beta_{2,2}$) o distribuirlo su entrambi ($\beta_{1,2}$).

- μ: E' la densità di massa dell'oggetto e partecipa attivamente nella definizione dell'energia cinetica.

- γ: E' la densità di dissipazione: una valore diverso da 0 tende, col passare del tempo, a portare il sistema in uno stato stabile dissipando l'energia cinetica. Un valore uguale a 0 lascia perpetuare il moto all'infinito in quanto l'energia passa di continuo da cinetica a potenziale senza attenuazioni.

- *forze in X, Y, Z*: Sono semplicemente forze applicate all'intera geometria, lungo l'asse relativo.

Al lato pratico, questi parametri possono essere inseriti in Iado come spiegato meglio nel prossimo capitolo, e la simulazione prodotta avviene come segue:

1. Il potenziale elastico, dipendente dai parametri inseriti, si traduce in energia cinetica.

2. Questa energia, ridotta da una eventuale forza dissipativa, muove i controlpoints della geometria.

3. L'energia cinetica "ricarica" quella potenziale, perpetuando il moto finché l'eventuale densità di dissipazione non porta il sistema in uno stato di quiete, oppure, per $\gamma = 0$, all'infinito.

Capitolo 2

IADO: Interactive Animation with Dnurbs and OpenGL

2.1 Storia

IADO nasce nel 2003 nella versione 1.0 sviluppata in [PIGN03], con la funzione sostanziale di GUI per il pacchetto dnurbs, oltre che quella di strumento di visualizzazione di curve/superfici. Grazie a Iado la deformazione di nurbs basata sul modello fisico che offrono le Dynamic Nurbs, diventa interattiva. Lo stesso autore, poco dopo rilascia la versione 1.1, dove in sostanza viene aggiunta la possibilità di lavorare con più curve/superfici contemporaneamente nello stesso ambiente. Con il lavoro di Ciccone, [CICC03], si giunge alla versione 1.2, dove vengono corretti alcuni bugs rilevati e migliorati alcuni aspetti dell'interfaccia per rendere più comodo il lavoro dell'utente. Le modifiche e gli adattamenti apportati da me (descritti nel Capitolo 3) portano il software alla versione 1.3.

2.2 Dipendenze

2.2.1 OpenGL

IADO si appoggia massicciamente in ogni suo aspetto alla libreria grafica OpenGL, che come sappiamo rappresenta la principale interfaccia all'hardware grafico per gli ambienti Unix/XWindow. OpenGL (Open Graphics Library) è una specifica che definisce una API per più linguaggi e per più piattaforme per scrivere applicazioni che producono grafica 2D e 3D. L'interfaccia consiste in circa 250 diverse chiamate di funzione che si possono usare per disegnare complesse scene tridimensionali a partire da semplici primitive.

2.2.2 GLU

Consiste in un numero di funzioni che utilizzano la libreria di base OpenGL per fornire routine di alto livello per il disegno, partento dalle routine più primitive che fornisce. E' di solito è distribuito con il pacchetto base di OpenGL. Tra queste caratteristiche ci sono la generazione di texture mipmap, la rappresentazione di NURBS, la tessellation di primitive poligonali, l'interpretazione dei codici di errore OpenGL, una gamma ampia di routine per il posizionamento facilitato della vista, generalmente più human-friendly rispetto alle primitive presentate da OpenGL. Fornisce inoltre primitive aggiuntive per il disegno semplificato di sfere, cilindri e dischi.

2.2.3 GLUT

GLUT è l'abbreviazione di OpenGL Utility Toolkit, un toolkit indipendente dal sistema operativo e dal windows manager per scrivere programmi in OpenGL. Questa libreria ci fornisce delle utili primitive per gestione di un altro aspetto fondamentale del progetto, e cioè la gestione delle finestre. La creazione delle stesse, la gestione di menu, il disegno di font, la gestione

di input da tastiera/mouse, il disegno automatico di geometrie relativamente complesse (teapots, coni, ...) sono alcune delle comode funzionalità che offre, ampiamente utilizzate nel pacchetto.

2.2.4 GLUI

Non è nient'altro che una interfaccia costruita sulle stesse GLUT di cui ne espande le funzionalità, e ci fornisce una ulteriore astrazione per gestione di finestre. Fornisce primitive per la creazione immediata di pulsanti, checkboxes, menu a tendina, box di testo per l'immissione di testo, radiobutton, spinners, controlli automatizzati per la traslazione e la scala di oggetti nello spazio di lavoro, tutti supporti usati per dare una interfaccia utente il più user-friendly possibile. Da questa versione di Iado, è necessario installare questa libreria sul proprio sistema operativo in quanto è stato abbandonato l'uso della vecchia libreria statica.

2.2.5 FFmpeg

FFmpeg è una suite software completa per registrare, convertire e riprodurre audio e video. Si basa su *libavcodec*, libreria per la codifica audio/video. FFmpeg è sviluppato su Linux, ma può essere compilato ed eseguito su qualunque dei principali sistemi operativi, incluso Microsoft Windows. Le animazioni realizzate in seguito ad una simulazione possono essere esportate in formato video, e la FFmpeg ci fornisce proprio le primitive necessarie ad un video encoding. Nel sorgente che svolge questa operazione non si fa uso dell'intera suite, ma solamente della *libavcodec* e della *libavformat*. Come per le glui, è necessario installare queste due librerie sul proprio sistema operativo, in quanto la vecchia FFmpeg 0.4.7 statica è stata abbandonata. A seguito di questo, per permetterne il corretto funzionamento, l'intero codice sorgente

di *movie.c* (che permette a Iado di effettuare l'encoding video) è stato riscritto.

2.3 Come si presenta

All'avvio di Iado ci troveremo davanti ad una finestra più grande che partirà da coordinate 0,0 (in alto a sinistra nello schermo) la cui funzione sarà quella di visualizzare le nostre curve/superfici, e alla sua destra una finestra più piccola che ci permetterà di iniziare ad utilizzare le funzionalità che offre il pacchetto, caricando una nurbs o una animazione. Lanciando Iado senza passare argomenti da riga di comando la finestra principale che appararirà avrà la dimensione di default, 800x600. Per personalizzarne la grandezza, occorre passare 2 argomenti: larghezza e altezza.

Nella figura sottostante è dato un esempio di come si presenta Iado all'avvio:

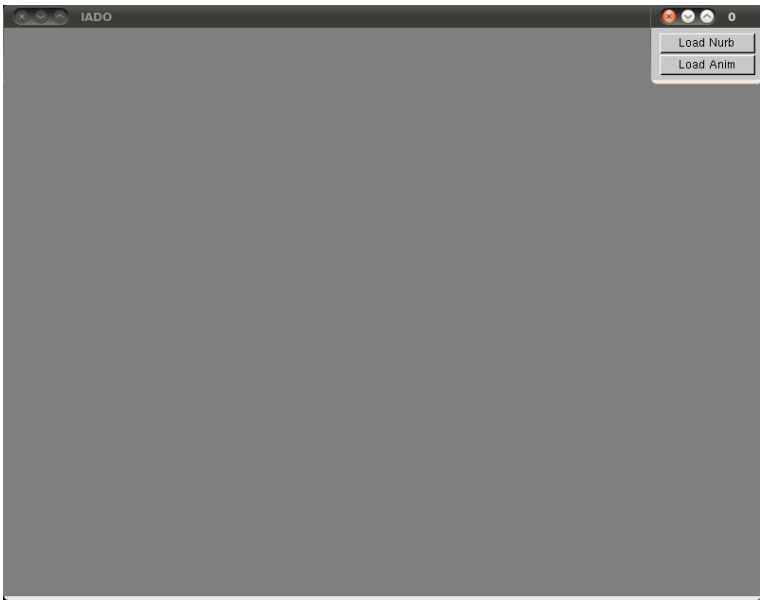

Figura 2.1: Iado: schermata iniziale

2.3.1 Caricamento di curve, superfici o animazioni

Con il pulsante load nurbs è possibile caricare una curva o una superficie:

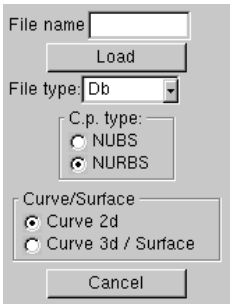

Figura 2.2: Iado: caricamento di nurbs o animazioni

I formati accettati sono due, *.db* e *.bin* . Il primo contiene una parte atta al salvataggio dei parametri fisici e di calcolo delle nurbs: in pratica supporta le Dnurbs. Il secondo invece non prevede questi campi e riveste quindi un ruolo di secondo piano nell'analisi specifica di IADO. La directory alla quale Iado farà riferimento in questa fase sarà */curves* per il caricamento di curve, e */surfaces* per le superfici. Dopo aver selezionato il tipo di file che si vuole caricare, occorre inserire nella textbox il nome del file completo di estensione ma senza percorso, e selezionare nei due radiobutton sottostanti se si desidera caricare una NURBS o una NUBS (tutti i knot con peso unitario), e se il file in questione descrive una curva o una superficie. Iado, come detto in precedenza, andrà a cercare il nome del file nella cartella */curves* se abbiamo selezionato dal radiobox il caricamento di una curva2d, nella cartella */surfaces* se abbiamo scelto una superficie. Nel caso che il file che si vuole loadare non corrisponda ai parametri appena inseriti (o che quel nomefile non esista), riceveremo un messaggio di errore. In caso positivo invece nella finestra principale comparirà la rappresentazione grafica della nostra curva/superficie e sulla parte destra appariranno alcune nuove finestre che ci permetterenno di eseguire tutte le funzioni.

Con il pulsante "*load anim*" sarà invece possibile caricare una animazione già costruita con Iado ,ad esempio in seguito ad una simulazione di una dnurbs. La seguente immagine mosta la finestra di caricamento di una animazione:

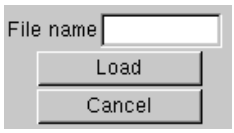

Figura 2.3: Iado: caricamento di animazioni

Il file dovrà essere nel formato *.anm*. Se inseriremo un nomefile esistente e corrispondente al tipo di formato richiesto, il primo frame della nostra animazione apparirà nella finestra principale. Le finestre che si apriranno, a caricamento avvenuto, saranno le stesse descritte prima per la funzionalità "*load nurbs*", tranne la "*dnurbs active par*" e la "*simulation*", in quanto queste due offrono funzionalità di cui non è possibile usufruire in questo momento.

2.4 Dettaglio di ogni finestra

Le finestre che si apriranno a caricamento correttamente effettuato saranno:

2.4.1 Cp movement and nurbs mod

Questa finestra consente la modifica della vista o della geometria dell'oggetto:

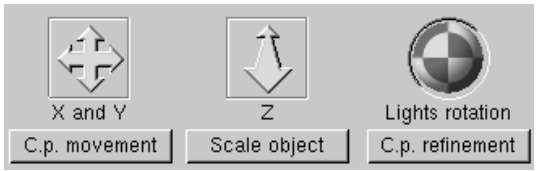

Figura 2.4: Iado: modifica della vista o della geometria

Tramite i button "x e y" e *"z"* è possibile, cliccandoci sopra e trascinando, muovere la direzione di vista lungo i rispettivi assi, per la migliore visualizzazione possibile dell'oggetto visualizzato all'interno della finestra principale. Il button *"light rotation"* ci permette di ruotare appunto la posizione della luce direzionale usata nella resa 3d della nostra nurbs, in modo da non lasciare l'oggetto in ombra. Il primo bottone della seconda riga, *"CP Movement"*, lascierà visualizzati nella nostra finestra principale solo i controlpoints della nostra curva/superficie. Potranno poi essere selezionati con il tasto destro (a selezione avvenuta diventerà giallo) e mossi con i bottoni apparsi nella finestra sottostante, rispettivamente in *"x e y"*, *"z"* o ruotati, sempre cliccando sul corrispettivo pulsante e trascinando col mouse nella direzione desiderata. Il secondo bottone ci permette comodamente di effettuare una scala del nostro oggetto: alla sua pressione si aprirà una finestra che chiederà mediante 3 checkbox su quali assi applicare la scala (x,y,z), con una textbox quanto sarà il coefficiente di scala per ogni step, e il solito pulsante da trascinare col mouse, avanti per ingrandire e indietro per diminuire le dimensioni. Il terzo bottone, il "*cp refinement*", ci dà la possibilità di aggiungere punti di controllo alla nostra curva/superficie in maniera automatica, semplicemente con la pressione di questo pulsante. Sono proprio i controlpoints ad essere mossi sotto l'effetto di forze applicate o di tensioni locali date da energia elastica: di conseguenza una geometria ricca di controlpoints approssimerà la sua evoluzione in maniera più precisa e fedele alla realtà, anche se ne aumenta in maniera direttamente proporzionale la complessità di calcolo.

2.4.2 Active curv/surf

Lo stesso creatore di Iado, con la versione 1.1 ha aggiunto possibilità all'utente di lavorare con più curve/superfici contemporaneamente.

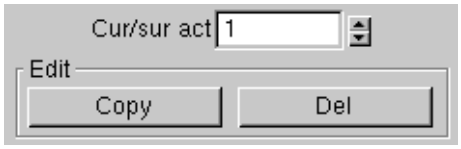

Figura 2.5: Iado: selezione della nurbs attiva

Tramite questa finestra si ha la possibilità, tramite lo spinner posizionato nella prima riga, di selezionare la curva/superficie attiva in quel dato istante, per permetterne ad esempio la modifica dei parametri di simulazione. I due bottoni nella seconda riga ci permettono infine di copiare o eliminare l'oggetto selezionato come attivo nel passo precedente.

2.4.3 Options

La finestra "*options*" offre diverse opzioni di controllo della visualizzazione del nostro oggetto.

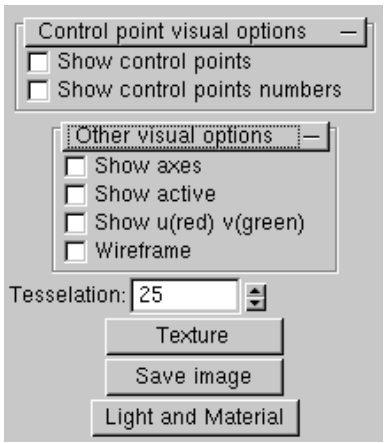

Figura 2.6: Iado: opzioni di visualizzazione

Nella parte superiore sono presenti due menù a tendina: espandendo il primo ("Control point visual options") si presenteranno due chechboxes che ci offriranno la possibilità rispettivamente di rendere visibili i controlpoints della nostra nurbs e di far appararire al fianco di ognuno di essi il suo numero identificativo: quest'ultima feature è utile nel caso scegliessimo di vincolare alcuni controlpoints in maniera testuale, dove viene appunto richiesto il numero identificativo di quelli su cui si desidera operare. Espandendo il secondo menù a tendina avremo a disposizione quattro checkbox:

- "*show axes*"

  Ci permette di ottenere una rappresentazione grafica degli assi cartesiani per capire l'orientamento del nostro oggetto (funzione molto utile al fine di applicare correttamente le forze nella giusta direzione).

- "*show active*"

  Ci fa capire immediatamente, mediante una geometria blu lampeggiante, quale delle curve/superfici è attiva in quel dato momento, utile sopratutto quando si lavora con più oggetti nello stesso ambiente.

- "*show u e v*"

  Mostra graficamente in rosso le curve che compongono il nostro oggetto in u, in verde quelle in v.

- *show wireframe*

  Mostra la nostra superficie in forma wireframe anzichè renderizzata.

Lo spinner sottostante ci dà invece la possibilità di impostare in tempo reale la tessellation della nostra geometria 3d: valori bassi significano maggior dettaglio nel disegno della nostra superficie, valori alti minor dettaglio. Il button sottostante, *texture*, come dice la parola ci permette di applicare una texture alla nostra superficie. Le immagini possono essere importate in formato bitmap (bmp) o targa (tga), scegliendo la rispettiva estensione dal menu a tendina della finestra che appararirà. Non occorre inserire nessun percorso

nella textbox ma solo il nome del file comprensivo di estensione: Iado andrà automaticamente a cercarlo dentro alla cartella */textures* e, se i campi indicheranno un file valido applicherà la texture, al contrario ci darà un messaggio di errore. Da questa versione, la 1.3, l'applicazione della texture avviene in modalità *clamp_to_edge* anzichè *repeat*. Il pulsante "*Save Image*" sottostante permette invece di salvare in una immagine di formato bitmap o targa la scena visualizzata nella finestra principale in quel momento, grazie ad un altra window che si aprirà alla pressione del pulsante. Anche in questo caso non è necessario inserire il percorso all'interno della textbox, l'immagine verrà salvata nella directory */images*. L'ultimo button, "*light and material*", aggiunto con lo sviluppo alla versione 1.2 di [CICC03], ci dà la possibilità di modificare gli attributi del materiale che costituisce il nostro poligono e della luce che illumina la scena, divisi per componenti rgb e alfa.

2.4.4 Dnurbs active par

Questa è la finestra chiave si Iado, in quanto permette di settare i parametri che, passati al modello fisico implementato in dnurbs assieme alla nostra nurbs di partenza, ci permettono di effettuare la nostra simulazione pseudo-realistica (fig. 2.7). Abbiamo una serie di button, il cui numero varia nel caso si sia caricata una curva o una superficie, che ci permettono di impostare tutti i parametri necessari alla simulazione: densità di massa, densità di dissipazione dell'energia, coefficienti di tensioni locali e rigidità locali, forza applicata in x, y, z. Tramite questi button è possibile specificare una funzione associata al parametro scelto, che può essere espressa scegliendo fra *spline* o *polinomiale,* con relativo grado in u e v. I coefficienti relativi alla funzione associata possono essere inseriti in maniera testuale o grafica: nella prima modalità viene semplicemente richiesto di inserire in una textbox il valore numerico corrispondente al coefficiente, nella seconda vengono visualizzati alcuni grafici e, tramite il mouse, è possibile modificare i valori dei singoli coefficienti. Sotto abbiamo due gruppi di radiobox, che ci permettono rispettivamente di vincolare tra loro i controlpoints sovrapposti (necessari nelle superfici chiuse) o di

lasciarli indipendenti, e di scegliere il tipo di vincolo applicato ai controlpoints che si è scelto di vincolare: geometrico o meccanico.

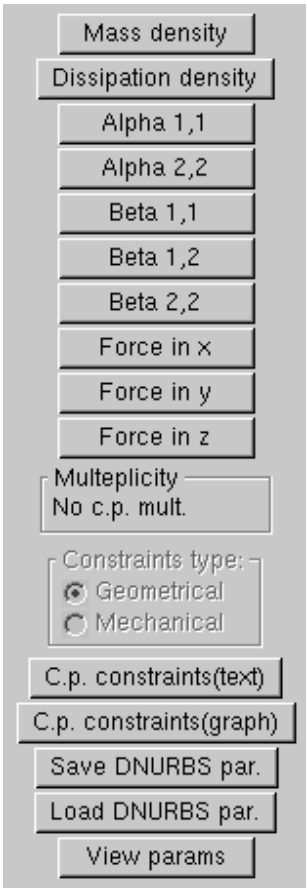

Figura 2.7: Iado: impostazione dei parametri fisici

Per maggiori informazioni riguardo a questi parametri si rimanda al capitolo primo. Nella parte di più bassa delle finestra abbiamo infine altri cinque button. I primi due offrono offrono la possibilità di impostare quali controlpoints vincolare, rispettivamente in modalità grafica o testuale: nella primo caso, la rappresentazione della nostra nurbs sparirà lasciando spazio ai soli cp; con il mouse andremo a fare click-destro su quelli interessati, che diventeranno gialli. Il button "*select/deselect all*" ci permette di selezionarli o deselezionarli tutti in un colpo solo. Una volta terminato il procedimento di scelta dei con-

trolpoints, verrà richiesto se vincolare tutti quelli selezionati agli stessi assi o se dar la possibilità di impostarne uno alla volta. Fatta questa scelta apparirà una ulteriore finestra il cui scopo è quello di farci scegliere gli assi ai quali vogliamo associare il vincolo: una volta sola se abbiamo scelto di vincolarli tutti nello stesso modo, una volta per ogni controlpoint selezionato nell'altro caso. I due button sottostanti ci danno la possibilità di effettuare load o save all'interno di un file dei parametri che abbiamo appena impostato; la directory alla quale Iado farà riferimento sia in fase di caricamento che di salvataggio sarà */par_curves* se abbiamo a che fare con una curva, */par_surfaces* se abbiamo a che fare con una superficie. Infine l'ultimo button ci apre una window che dà un comodo riassunto di tutti i coefficienti dei parametri appena elencati, applicati in quel momento. Da notare il fatto che *dncurv/dnsurf* necessita che tutti questi parametri vengano inseriti; in caso contrario, in fase di avvio della simulazione, riceveremo un messaggio di errore. Questa finestra è attiva solamente quando non è in riproduzione nessun'altra animazione prodotta da una simulazione o caricata da un file.

2.4.5 Simulation

E' la finestra tramite la quale possiamo applicare i parametri del modello fisico precedentemente inseriti alla nostra nurbs e produrre una rappresentazione grafica della sua evoluzione nel tempo.

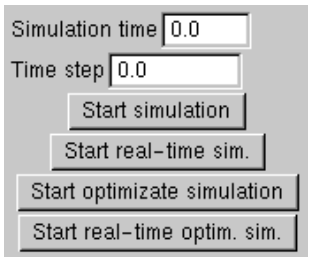

Figura 2.8: Iado: finestra di simulazione

Abbiamo la possibilità di inserire tramite textbox la durata totale della nostra run di simulazione e il timestep. Una volta impostati correttamente questi due valori, con la pressione di uno dei button sottostanti si fa partire la run. I tipi di simulazione si dividono in due parametri da scegliere, premendo il relativo bottone: simulazione in real-time o calcolata per intero in un solo passaggio, simulazione classica od ottimizzata. Questa ultima scelta è possibile solamente se stiamo per eseguire la simulazione di una superficie. Se non sono stati inseriti tutti i parametri fisici tramite l'interfaccia contenuta nella finestra descritta nel punto precedente, la simulazione non andrà a buon fine e riceveremo un messaggio di errore.

Tipi di simulazione

- *Classico*

  In questo tipo di calcolo l'utente ha la possibilità di visualizzare l'animazione risultante solo quando tutte le curve/superfici che compongono ogni frame dell'animazione stessa sono state calcolate. Il numero di queste superfici o curve dipende in maniera diretta dal valore $sim\_time/time\_step$, valori che abbiamo inserito in fase di configurazione della simulazione, e di conseguenza anche il tempo di calcolo. A supporto di questa fase, nella versione 1.3 di Iado è stata inserita una progressbar che ci darà un feedback visivo sul fatto che il calcolo dell'evoluzione del sistema stia realmente procedendo e a che punto si trova. Il procedimento avviene serialmente, in quanto non è possibile calcolare lo stato del sistema all'istante $n+1$ se non è ancora terminato il calcolo in n (mi serve sapere come è evoluta la mia nurbs nel frattempo).

- *Ottimizzato*

  Questo tipo di calcolo ha lo scopo di ridurre i tempi di elaborazione cercando un modo più furbo di calcolare le superfici risultanti dalla nostra simulazione, e per farlo si appoggia sul concetto di *skinning*, un processo secondo il quale da un insieme di curve-sezione, interpolandole, è pos-

sibile formare una superficie. La simulazione ottimizzata sfrutta questo concetto al contrario: una superficie può essere suddivisa in un certo numero di curve-sezione che ne descrivono la geometria, posso iterare il modello fisico delle *dynamic nurbs* offerto da *dncurv* a queste curve (procedimento che è nettamente più rapido rispetto al calcolo classico dello stesso oggetto sottoposto ai medesimi parametri fisici), e dalle curve risultanti ricostruirmi la superficie tramite il processo di *skinning*. Entrando nel dettaglio del costo computazionale, considerando n la dimensione della matrice del control points, un calcolo di una animazione di una superficie è dell'ordine di $O(n^2)$, mentre la risoluzione del calcolo dell'animazione di una curva comporta un calcolo dell'ordine di $O(n)$. Iado suddivide le curve in u e in v, e le passa a *dncurv* per l'elaborazione. Il calcolo di ogni curva che compone la nostra superficie viene iterato parallelamente, il che comporta una ulteriore velocizzazione nei tempi.

Il calcolo ottimizzato, così come implementato in Iado, purtroppo non dà garanzia di ottenere sempre buoni risultati: per definizione, infatti, la procedura di divisione in curve, per essere corretta, dovrebbe costruire un insieme di curve che effettivamente siano sezioni della superficie ad un dato intervallo. Attualmente questo non viene propriamente effettuato, ma vengono prese in considerazione come curve le colonne o righe della matrice di controlpoints; dopo il calcolo queste tornano ad essere le nuove righe/colonne della nuova superficie generata. Alcune volte questo processo approssima correttamente quello che la definizione dice, ma la cosa non è sempre verificata e alle volte i risultati sono completamente diversi da quelli aspettati. Dalla *proprietà di modifica locale delle NURBS*, che cpme muovendo un controlpoints $P_{i,j}$ si modificherà la geometria nel rettangolo $[u_i, u_{i+p+1}) \times [v_j, v_{j+p+1})$, si può capire che una sezione non può essere definita solo da una riga i di controlpoints, ma anche dalle $p+1$ adiacenti.

- *Calcolo in tempo reale*

Selezionando una simulazione operante in questa modalità, l'utente ha la possibilità di visualizzare come evolve l'animazione "step by step", senza attenderne il calcolo completo: in altre parole c'è la possibilità di vedere ogni frame generato dalla simulazione appena il suo calcolo è terminato. I pro di questa modalità sono facilmente individuabili in una maggiore flessibilità: qual'ora ci accorgessimo che i risultati della simulazione stanno vertendo verso una direzione inaspettata, possiamo subito stoppare, fare rewind e effettuare una nuova simulazione apportando le opportune modifiche. Il "contro" più incisivo è l'aumento importante del tempo di calcolo dell'intera run di simulazione. Per come è organizzato concettualmente questo tipo di iterazione, è necessario infatti ricalcolare tutti i passi n di evoluzione del sistema ogni volta che si desidera effettuare la simulazione dello step $n + 1$. Da notare il fatto che scegliendo il calcolo real-time non è necessario inserire il valore di *simulation time*, in quanto, la simulazione proseguirà di step in step fino all'interruzione da parte dell'utente mediante il tasto stop.

La finestra "*simulation*" è attiva solamente quando non è in riproduzione nessun'altra animazione prodotta da una simulazione o caricata da un file.

2.4.6 Nurbs animation

Quando *dncurv/dnsurf* termina il calcolo della simulazione dell'evoluzione del sistema secondo il modello fisico, Iado apre una finestra che ci permette di interagire in diversi modi con l'animazione prodotta.

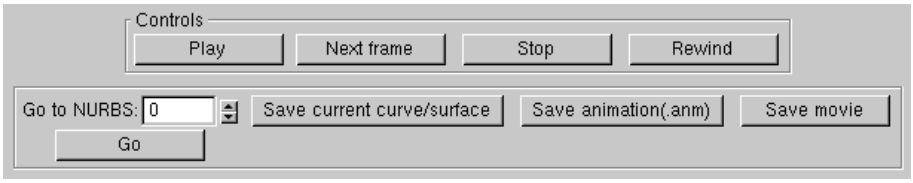

Figura 2.9: Iado: finestra di animazione

Tramite il button *Play* possiamo guardarla nella sua interezza (se si è usato il metodo real-time, questo bottone farà partire la simulazione dei vari step che proseguirà fino alla pressione dello *Stop*), il pulsante *Stop* metterà in pausa la riproduzione (in caso di simulazione real-time farà cessare il processo simulativo stesso), infine *Rewind* consente di tornare al primo frame. I controlli nella seconda riga permettono rispettivamente di saltare ad un frame prestabilito, di salvare la curva/superficie attiva e visualizzata in quel momento nella finestra principale o salvare l'intera animazione in formato *.anm*, per essere ad esempio caricata e riprodotta in un secondo momento. Infine l'ultimo pulsante consente di avviare la creazione di un movie e alla sua pressione ci troveremo davanti a un'altra finestra che ci consentirà di impostare alcuni parametri:

- *bitrate:* direttamente proporzionale alla qualità, rappresenta il numero di bit che compongono ogni secondo del video che si vuole produrre.
- *frame rate:* quanti frame sono inseriti in ogni secondo di riproduzione.
- *frame skip:* il codec video salterà un frame ogni n, dove n è il valore che assegneremo a questo parametro.
- *tempo:* durata in secondi totale del video prodotto.
- *formato:* è possibile scegliere 2 formati. MPEG1 e MPEG4(divx).

Nella texbox è necessario inserire solamente il *nomefile*, privo quindi sia di estensione che di percorso. Alla pressione del tasto salva partirà l'encoding video: da questa versione del software il procedimento supportato da una progress bar che ci dirà lo stato di avanzamento dell'operazione. Iado posizionerà il movie nella cartella */movies*, con estensione *.mpg* se si è scelto il formato MPEG1, *.m4v* nel caso sia sia scelto DIVX.

C'è in conclusione da precisare che la pressione del button "*save movie*" avvia l'encoding del video a partire dal frame che risulta attivo in quel momento: se si desidera quindi effettuare un video che parta dall'inizio della nostra

simulazione, occorre prima portarsi al primo frame dell'animazione prodotta, utilizzando i metodi sopra descritti, e poi premere il button per la creazione del video. Dalla questa versione del software (1.3) la riproduzione di una animazione disabilita automaticamente una serie di finestre, il cui utilizzo in questa fase risultava insensato e portava a comportamenti anomali del software stesso: per effettuare dunque nuove operazioni come ad esempio una modifica di parametri fisici e avvio di una nuova simulazione, occorre portarsi al primo frame con il tasto Rewind o con lo spinner apposito. Inoltre, visto l'abbandono della vecchia libreria FFmpeg 0.4.7 statica, tutto il sorgente c riguardante la creazione del video è stato rivisto per adattarsi alle nuove primitive offerte dalle nuove FFmpeg.

Capitolo 3

IADO: “alzati e cammina!”

In questo terzo capitolo verranno descritte tutte le modifiche effettuate a livello di codice al pacchetto Iado, che lo hanno portato alla versione attuale, la 1.3. Come detto nella parte introduttiva, il motivo principale per cui si è scelto di effettuare il lavoro di “riparazione e miglioramento” di questo pacchetto, non più mantenuto da nove anni, è il grande interesse riposto verso il mondo delle DNURBS da parte della Computer Graphics in questo momento. La prima parte tratterà delle scelte e dei processi che sono stati ritenuti necessari per la “rimessa in funzione” del pacchetto: saranno quindi esposti i vari problemi incontrati sia in fase di compilazione che a runtime, fornendo una spiegazione ad ogni soluzione scelta. Nella seconda parte verranno dettagliate invece tutte le modifiche atte al miglioramento del software e alla comodità di utilizzo da parte dell’utente, la cui necessità è nata da un utilizzo ripetuto delle funzionalità offerte dell’interfaccia grafica. Verranno messe anche in evidenza la parti di codice che hanno richiesto un *refactoring* per adattarsi all’upgrade delle librerie su cui si appoggiavano.

3.1 “Riparazione” del pacchetto

Lo sviluppo ed il mantenimento di *Iado* e della libreria *dncurv/dnsurf* a cui si appoggia per il calcolo dell’evoluzione del sistema, è purtroppo fermo dal

2003. In [RUBI96], l'autore creò il motore di calcolo dncurv/dnsurf, nel 2003 con [PIGN03] veniva creata la prima versione di Iado, la 1.0, con lo scopo di fornire un frontend al lavoro svolto in [RUBI96]. Lo stesso autore creava poi nello stesso anno la versione 1.1 aggiungendo alcune funzionalità. Nello stesso anno, con [CICC03] venivano risolti alcuni fastidiosi bug e migliorati alcuni aspetti dell'interfaccia. Entrambi i pacchetti sono scritti totalmente in *C++*, fatta eccezione per il solo sorgente di Iado *movie.c*, scritto appunto in C puro. Sono nove anni che il software non subisce alcuna modifica/aggiornamento, e come pronosticato, il primo approccio alla fase di compilazione non si è risolto positivamente.

3.1.1 Iado non funzionante in fase di compilazione

Le cause principali alla base del fatto che Iado non risultasse correttamente compilabile sulla macchina utilizzata, che utilizza la versione *4.4.3* di GCC, erano due:

- alcuni aspetti dello standard-c nel frattempo sono cambiati. Basti pensare che dal 2003 ad oggi, *GCC* (GNU Compiler Collection, il compilatore multi-target utilizzato per la compilazione di Iado e dncurv/dnsurf), ha visto lo sviluppo di ben 50 releases, dalla *3.3.2* alla *4.6.2* disponibile ora.

- alcune parti del codice si appoggiavano a primitive fornite da librerie statiche obsolete e non più attualmente compilabili.

Il primo problema è stato superato agevolmente con una correzione prettamente sintattica di tutto il codice, che lo rendesse "comprensibile" al compilatore attualmente utilizzato e allineato ai nuovi standard imposti.

Nella risoluzione del secondo aspetto, sono state inizialmente valutate due possibilità: l'eliminazione di ogni vecchia libreria statica a favore del riferimento a librerie dinamiche oppure la loro bonifica per consentirne nuovamente la compilazione e quindi l'utilizzo. Dopo aver percorso per qualche tempo entrambe le strade, ho optato per la prima in quanto volevo modernizzare il

codice adattandolo alle nuove librerie. Mentre per la GLUI (una delle due lib statiche) non ci sono stati particolari problemi, più complesso è stato l'adattamento del codice alle nuove *libavformat* e *libavcodec* (prima contenute nelle FFmpeg ver. 0.4.7) in quanto molte chiamate a funzioni dichiarate nelle vecchie librerie statiche sono deprecate da anni e sono state sostituite con altre (che operano in modo spesso molto diverso). E' stato quindi più conveniente riscrivere il codice sorgente di *movie.c*, insieme di routine che si occupa dell'encoding del video. Al termine di queste operazioni la compilazione è stata effettuata con successo, ed si è proceduto con l'eliminazione della causa di tutti i warning, attivando in fase di compilazione i flag *-Wall -Wextra*, per una visualizzazione completa di ogni anomalia che il compilatore incontri, spesso causa di dimenticanze da parte del programmatore che evolvono in bug a runtime.

3.1.2 dncurv/dnsurf non funzionante in fase di compilazione

Anche per quanto riguarda il sorgente del pacchetto *dncurv/dnsurf*, come detto in precedenza, l'ultimo sviluppo non è recente e la compilazione falliva. Il primo passo, anche in questo caso, è stato quello di effettuare una correzione prettamente sintattica del codice atta ad allinearlo ai nuovi standard, in quanto in caso contrario il compilatore non saprebbe interpretare quel codice. Le parti che hanno richiesto più attenzione sono stati i template, dove l'autore si crea i propri tipi di dato (double linked list, vettore e matrice), che usa poi assiduamente per implementare e calcolare le equazioni differenziali alla base del modello fisico delle *DNURBS*. Con l'organizzazione del codice all'interno di direttori dedicati e la correzione del makefile, l'operazione si concludeva a anche dnurbs compilava correttamente.

3.1.3 Iado non funzionante a runtime

Al termine delle operazioni appena descritte, sono stati correttamente ottenuti gli eseguibili sia di *Iado*, che i due eseguibili per il calcolo dell'evoluzione DNURBS, *dncurv* e *dnsurf*. A questo punto è iniziata la fase di testing, che ha portato subito alla luce un aspetto: la fragilità del codice. Ripetuti crash del'intero sistema operativo prima, e del solo software poi, e comportamenti anomali che rendevano praticamente inutilizzabili molte funzionalità, sono stati molto frequenti nelle run di test; tutto ciò ha portato alla luce la priorità di cercare di *bonificare* il codice per renderlo pienamente operativo in tutto quello che offre, prima di qualsiasi altra operazione. Dopo una analisi del codice, i punti con maggiore criticità (e quindi probabilità di crash o comportamenti anomali) e le rispettive modalità di correzione, senza entrare troppo nella specifica sintassi di programmazione, sono stati individuati in:

- *Mancanza di controlli sull'accesso ad array statici spesso sottodimensionati.*

  Il codice abbondava di riferimenti ad array di dimensione fissa, senza che ci fosse alcun controllo sull'effettiva quantità di elementi che sarebbero poi andati a popolarlo. Questa anomalia era presente in tutti i casi in cui l'utente doveva inserire il nome di un file all'interno di una qualsiasi textbox, per il caricamento o il salvataggio di dati su di esso. Come è facilmente intuibile, questa è una situazione molto pericolosa in quanto non è prevedibile a priori quanto possa essere lunga una stringa contenente un nomefile inserita dall'utilizzatore, e porta inevitabilmente a un Segmentation fault.

- *Esteso di variabili globali.*

  Sono spesso fonti di errori tendenzialmente difficili da individuare e complicano enormemente gli sviluppi futuri.

- *Scarso utilizzo di variabili "const" dove possibile.*

Le variabili di tipo “const” offrono invece una buona sicurezza anche nell'utilizzo globale.

- *Allocazione dinamica di memoria utilizzata senza verificarne l'effettiva riuscita.*

  L'allocazione dinamica della memoria era usata senza mai effettuare un controllo sulla sua effettiva buona riuscita: in caso negativo, con il proseguimento dell'esecuzione, il crash era assicurato. Non era poi sempre presente il richiamo alla primitiva *free()* necessaria alla liberazione della memoria precedentemente allocata dinamicamente di cui non abbiamo più bisogno, portando ad uno spreco inutile di memoria.

- *Mancanza di un effettivo controllo sul valore di ritorno delle funzioni e gestione delle eccezioni.*

  Come per il caso precedente, non verificare mai se il valore di ritorno delle funzioni richiamate è quello atteso, può portare a crash o comportamenti anomali in fase di esecuzione, non sempre di facile individuazione.

- *Mancato controllo della* non nullità *dei puntatori.*

  Utilizzare per sbaglio un puntatore nullo per puntare a un oggetto porta inevitabilmente al crash.

Il maggior numero di queste criticità erano contenute nella funzione *copiafile()* (all'interno di *writenur.cpp*), il che la rendeva la causa principale dei problemi sopra descritti. Alla luce di ciò ho optato per la completa riscrittura di questa routine, precedentemente limitata all'uso solo con file di testo, e ora generalizzata al fine di poter lavorare con qualsiasi tipo di file binario, per essere pronta ad eventuali sviluppi futuri del pacchetto.

3.2 Migliorie apportate

A software pienamente eseguibile in tutte le funzionalità offerte, si è passato ad una fase di testing massiva dell'interfaccia grafica, alla ricerca di possibili

scomodità o punti comunque perfezionabili. Si vuole sottolineare come nessuno dei problemi individuati comprometteva realmente il corretto funzionamento del pacchetto, ma alcune situazioni non erano controllate correttamente ed un utilizzo improprio poteva portare ad un comportamento anomalo (o nel peggiore dei casi a un crash). Alcune altre scomodità probabilmente erano state volutamente trascurate negli sviluppi passati probabilmente per dedicare tempo a ottimizzazioni riguardanti sezioni più critiche, e solo ora hanno trovato la priorità. Nell'elenco che segue si cercherà quindi di spiegare dove si è individuata una possibile funzionalità migliorabile, e cosa è stato fatto a livello pratico per cercare di sanarla.

- Il software era privo di un qualsiasi output che, nelle operazioni più onerose dal punto di vista del tempo di calcolo, comunicasse all'utilizzatore lo stato di avanzamento (o quantomeno il fatto che stesse procedendo) della routine richiamata. Il tempo necessario a ultimare la simulazione di un sistema o la creazione di un video è direttamente proporzionale al numero di frames che l'utente decide di creare (dato dal rapporto dei parametri simulation_time time_step); calcolando quindi molto accuratamente o per un lungo periodo di tempo una deformazione di curva/superfici capitava di dover attendere anche minuti senza avere nessun feed-back. Ora quando l'utente decide di lanciare una simulazione o di effettuare un encoding video di una simulazione precedentemente calcolata, il software mostrerà una "*progress*" bar che nel titolo evidenzierà l'operazione in corso in quel dato istante, e nel campo centrale la percentuale di avanzamento; ad attività conclusa la barra scompare automaticamente.

- Essendo il software organizzato in tante piccole finestre dove ognuna raggruppa un certo numero di funzionalità, capitava che rimanessero disponibili all'utente delle operazioni che in quel dato momento non fossero consentite: un esempio può essere la modifica dei parametri di simulazione durante la visualizzazione di una simulazione precedentemente calcolata. A volte, oltre che non fornire il risultato aspettato,

il richiamo di queste routine in situazioni non contemplate portava al crash del programma. Sono stati inibite quindi, in certi momenti durante l'utilizzo, le finestre che contenevano funzionalità che in quel dato momento non ha senso (o non è concesso dal software) utilizzare.

- E' stata sistemata la geometria delle finestre e dei relativi titoli, rendendole tutte visibili e non sovrapposte all'avvio di Iado.

- In Iado si ha principalmente a che fare con superfici relativamente contenute e poco articolate: si è quindi deciso di modificare il metodo di applicazione delle texture da GL_REPEAT (ripetizione della texture fino a riempire l'area) a GL_CLAMP_TO_EDGE, così da espandere la texture per tutta la superficie riempendola in maniera più estetica.

- Il codice è stato organizzato in direttori; nella root del progetto sono presenti i sorgenti *.cpp* e *.c* ,e il *Makefile*, oltre agli eventuali eseguibili generati da quest'ultimo; nella direttorio *h* sono stati inseriti tutti i file di header, e in *e* sono stati creati ove necessario dei file contenenti le sole variabili *extern* dei rispettivi sorgenti c++ o c. Il procedimento ha reso ovviamente necessario anche un adattamento a livello di codice. La nuova organizzazione rende più immediato l'accesso alle parti interessate del sorgente e quindi agevola eventuali sviluppi futuri.

3.3 Installazione

In questa sezione verranno descritti i semplici passi necessari per compilare i sorgenti necessari ad ottenere gli eseguibili che compongono il pacchetto.

Il pacchetto al momento è formato da tre programmi distinti:

- *Iado*: il front-end, i cui sorgenti sono ospitati nella cartella */sources*

- *dncurv/dnsurf*: il motore di calcolo DNURBS, i cui sorgenti sono ospitati nella cartella */sources/dnurbs*

- *xprogress*: la progress-bar usata in fase di simulazione-encoding, i cui sorgenti sono ospitati nella cartella */sources/xprogress*

E' scontato dire che per il corretto funzionamento del pacchetto è necessario compilare correttamente tutti e tre i software, in quanto il codice di Iado andrà a richiamare gli altri due programmi esterni; in ogni directory dei rispettivi pacchetti, riportate sopra, è presente un makefile che permette rapidamente di effettuare questa operazione. La compilazione del sorgente di Iado produrrà l'eseguibile, chiamato appunto "*Iado*", nella root del pacchetto, che è quello da lanciare per iniziare ad utilizzarlo.

E' bene fare anche chiarezza sull'organizzazione delle directory presenti, per cui ne fornisco una rapida descrizione:

- */anim*: è la cartella dove vengono salvate le animazioni prodotte, e di conseguenza da dove vengono caricate in fase di load.

- */curves* e */surfaces*: da qui vengono caricate rispettivamente le curve e le superfici, tramite il pulsante "Load NURBS". Una valida soluzione per creare curve e superfici può essere l'uso di *xccurv* o *xcsurf*, due software della suite [XCMODEL], sempre creato dall'Università di Bologna.

- */images*: dove vengono salvate le immagini tramite il pulsante "Save image" ( )

- */movies*: in questo direttorio vengono salvati i video che si possono produrre in seguito ad una simulazione eseguita.

- */par_curves* e */par_surfaces*: qui vengono salvati i file contenenti i parametri di configurazione (fisici e di simulazione) rispettivamente per curve 2d e superfici 3d, e sempre da questa directory vengono caricati all'occorrenza.

- */textures*: qui bisogna porre le textures (in formato .bmp o .tga), in quanto in fase applicazione di textures, Iado andrà a cercare in questo direttorio.

Capitolo 4

Analisi sperimentale

La fase di sperimentazione e di analisi dei risultati ottenuti ha ricoperto una ruolo chiave in quanto, in un primo momento, ha permesso, utilizzando a fondo l'interfaccia di Iado, di capirne i punti migliorabili e, in un secondo, di avere un riscontro visivo di come i vari parametri partecipano all'evoluzione della geometria al variare del tempo.

In questo capitolo saranno quindi descritte alcune simulazioni effettuate, scelte tra le altre per far comprendere al meglio il significato che i vari parametri assumono all'interno della simulazione stessa.

Si avranno quindi diverse sottosezioni, una per ogni parametro principale, dove verrà richiamata brevemente la teoria per ricordarne la sua definizione e dove verranno forniti esempi di simulazione di curve e superfici che mostreranno visivamente il suo impatto sull'evoluzione della geometria.

A esempio è possibile osservare, applicando una forza, come si deformi una superficie sottile e piana a cui è stata data un certa elasticità e certi vincoli, simulando il comportamento realistico di una sorta di tappeto elastico. E' bene sottolineare nuovamente il fatto che queste simulazioni, seppur corrette dal punto di vista numerico, cerchino solamente di emulare il comportamento realistico che la nostra geometria avrebbe nel mondo reale: il modello in questione, come spiegato nel capitolo precedente, opera solamente sui controlpoints che sono le uniche entità sottoposte all'azione dei parame-

tri fisici, creando una sorta di approssimazione che si avvicinerà alla realtà all'aumentare del numero dei controlpoints stessi.

4.1 Vincoli

I vincoli rappresentano sicuramente il primo aspetto da studiare nella pianificazione di una simulazione. Senza la loro applicazione, in presenza di potenziale elastico o di rigidità, tutti i controlpoints sarebbero attratti e respinti senza sosta, facendo perdere completamente all'oggetto la sua forma, come nell'esempio sotto:

Parametro	tipo	valore
massa μ	polinomiale	20.0
elasticità α	polinomiale	2.0
rigidità β	polinomiale	0.01
timestep δ_t		0.02 sec
sim. time		1.8 sec

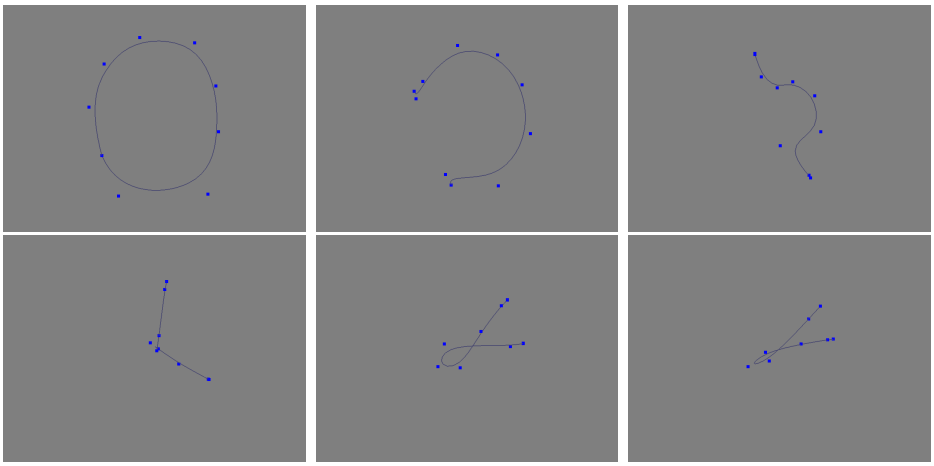

Figura 4.1: Cerchioide elastico non vincolato, visto ogni 0.2 sec.

Proprio perché il nostro scopo è quello di ottenere, tramite le nostre *run* di simulazione, una deformazione del nostro oggetto il più vicino possibile alla realtà, occorre impostare i giusti vincoli.

In presenza di forze esterne, invece, senza controlpoints vincolati, il corpo sarebbe solo semplicemente "traslato" all'interno dello spazio, uscendo probabilmente dal raggio visivo e producendo un risultato di scarso interesse. Occorre quindi farne un uso sapiente; in caso contrario si rischia di incappare in risultati molto diversi da quelli sperati. Nel caso di superfici chiuse, come il cerchioide in questione, occorre mantenere coincidenti i punti di giunzione tramite l'apposita funzione fornita da Iado (mantenimento molteplicità dei vincoli) per mantenere corretta la forma del corpo. Se, quindi, vincoliamo adeguatamente l'oggetto aumentiamo probabilmente le possibilità di rendere più realistica la simulazione. Ad esempio, vincolando ad entrambi gli assi x e y alcuni controlpoints del nostro cerchioide ed applicando gli stessi parametri, otterremo una deformazione più realistica, che mantiene anzitutto la curva chiusa e ci mostra molto più chiaramente come la forza elastica viene trasformata in cinetica spostando i controlpoints liberi (che a loro volta, in base al loro peso, deformano la curva).

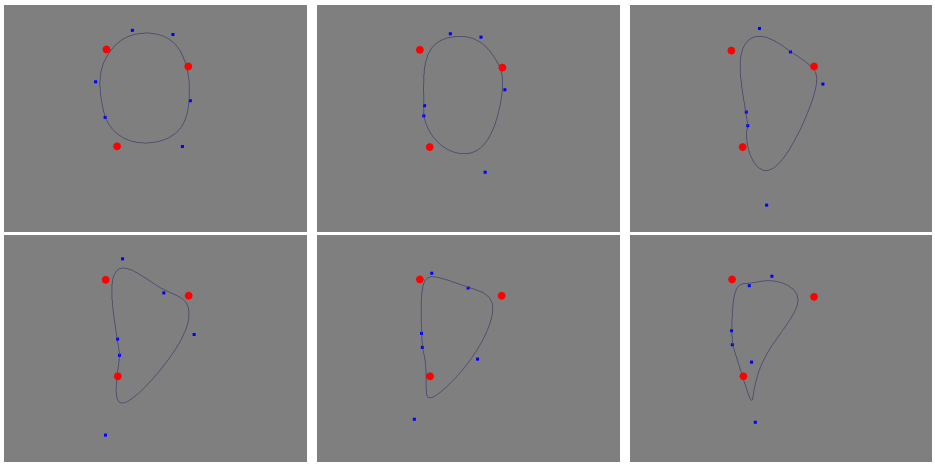

Figura 4.2: Cerchioide elastico vincolato, visto ogni 0.2 sec.

Questo esempio mostra proprio questo scenario, che non vuole simulare il comportamento di un qualche oggetto realistico ma solo mostrare come agiscono i vincoli applicati ad alcuni controlpoints. Nell'esempio fornito sono stati utilizzati vincoli su entrambi gli assi, ma è possibile vincolare controlpoints singolarmente sugli assi desiderati, per ottenere l'effetto voluto.

4.2 Elasticità e Rigidezza

Raggruppiamo questi due parametri in quanto sono strettamente correlati: mentre la prima cerca di avvicinare i controlpoints, la seconda cerca di mantenere rigido l'oggetto, opponendosi a qualsiasi forza compreso quindi l'elastica.

4.2.1 Parametro α

Il coefficiente di elasticità carica di energia potenziale la nostra geometria, che durante la fase di simulazione può trasformarsi in energia cinetica, avvicinando tra loro i controlpoints come se avessimo a che fare con un materiale elastico. Maggiore è il valore assegnato a questo parametro, e maggiore sarà l'elasticità del nostro materiale, richiedendo però anche più forza per "caricarlo".

Nell'esempio sopra si sono già visti gli effetti che la componente elastica ha sul movimento dei controlpoints, ma in questa sezione si cercherà, tramite l'uso di prove pratiche, di capire come varia la deformazione del nostro oggetto al variare appunto del coefficiente elastico. Si è deciso di fornire un esempio il più semplice possibile, partendo ad esempio da una geometria che vorrebbe simulare un elastico teso da un lato. Come prima prova vi si è poi applicato un coefficiente elastico molto basso per osservarne l'impatto:

Parametro	tipo	valore
massa μ	polinomiale	3.0
elasticità α	polinomiale	0.01
rigidità β	polinomiale	0.001
timestep δ_t		0.02 sec
sim. time		10.5 sec

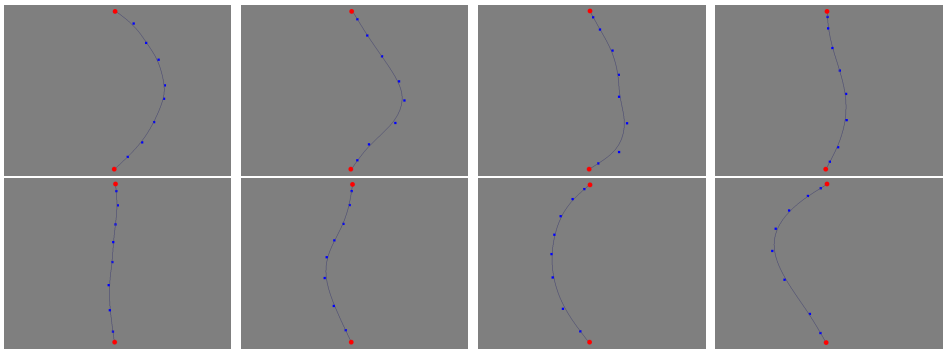

Figura 4.3: Elastico poco teso, visto ogni 1.5 sec.

Come si vede, il comportamento della curva appare "flaccido" in quanto la forza elastica prodotta dal modello fisico in queste condizioni è molto bassa. Chiaramente si tratta di un esempio limite, fornito solamente per visionare l'impatto di un coefficiente elastico molto basso. Se si alza considerevolmente il valore di elasticità, si assisterà ad un comportamento molto diverso della curva, che caricata di maggiore energia potenziale si deformerà in maniera più rapida e più uniforme.

Parametro	tipo	valore
massa μ	polinomiale	3.0
elasticità α	polinomiale	1
rigidità β	polinomiale	0.001
timestep δ_t		0.02 sec
sim. time		10.5 sec

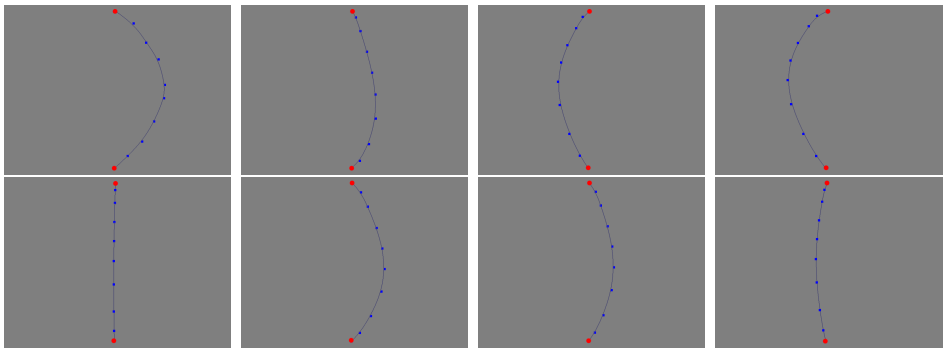

Figura 4.4: Elastico più teso, visto ogni 0.2 sec.

Da notare che ogni frame riportato in figura è stato prelevato ogni 0.2 secondi, a differenza degli 1.5 secondi dell'esempio precedente; si può quindi avere una idea di quanto più rapidamente si muova il corpo, spostato da una forza maggiore creata dalla maggiore densità elastica applicata.

4.2.2 Parametro β

Il coefficiente di rigidità β ha un ordine di grandezza inferiore rispetto alla tensione α: questo perché nel calcolo dell'energia potenziale U i parametri β sono legati alle derivate seconde delle funzioni base, mentre gli α alle derivate prime. Per questo motivo questi ultimi saranno moltiplicati per coefficienti più bassi; come spiegato ampiamento in [PIGN03], c'è quindi una forte predominanza dei parametri di rigidità rispetto a quelli di tensione. Nell'esempio sottostante viene portato all'estremo questo concetto, impostando un parametro di rigidezza estremamente elevato per fornire un esempio di come partecipa alla deformazione:

Parametro	tipo	valore
massa μ	polinomiale	3.0
elasticità α	polinomiale	0.0
rigidità β	polinomiale	1000
timestep δ_t		0.002 sec
sim. time		1.0 sec

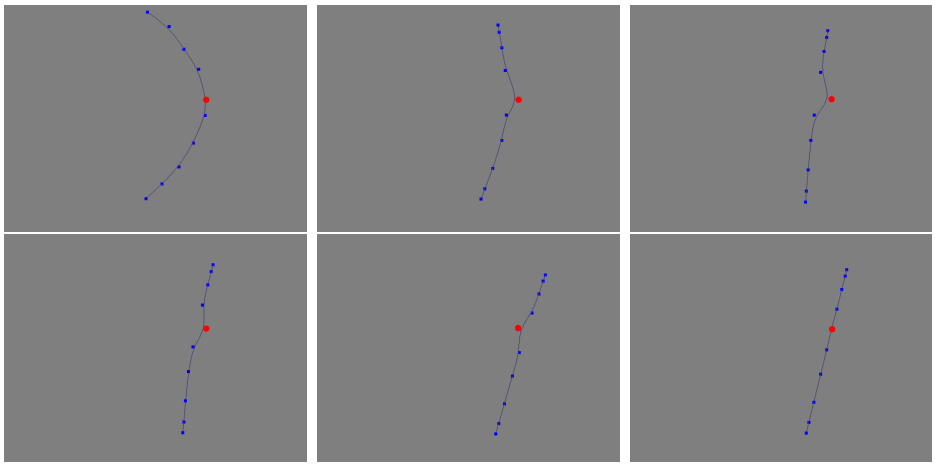

Figura 4.5: Asta rigida, vista ogni 0.002 sec.

Nel mondo reale ogni oggetto solido possiede una componente, seppur piccola di rigidità. Nel caso specifico del nostro modello fisico, essa partecipa quindi alla resa pseudo-realistica della deformazione. Valori elevati tenderanno a rendere l'oggetto molto rigido opponendosi alla forza elastica, bassi valori viceversa lo renderanno morbido. Partecipando quindi attivamente alla simulazione di un materiale elastico realistico, è sempre bene non lasciare valori uguali a 0 se lo scopo è quello di emulare un materiale esistente, come negli esempi precedenti.

4.3 Forze

La forza applicata al nostro oggetto è forse la componente più intuitiva del nostro modello fisico; si tratta di applicare, mediante una funzione spline o polinomiale, una forza definita da uno o più parametri costanti ad ogni controlpoints che compone il nostro oggetto. Questa volta effettueremo i test su una superficie 3d, per apprezzare risultati di deformazione oggetti che, seppur semplici per mantenere accettabili i tempi di calcolo, possiamo trovare nella realtà.

4.3.1 Forze deformanti

Prendiamo in considerazione un cilindroide privo di basi, al quale applichiamo una forza verso il basso (nel nostro caso asse y). Non sono ovviamente gestiti attriti o collisioni (in quanto come già detto più volte il modello opera solo su controlpoints), quindi per simulare il fatto che sia fissata da un lato (nel nostro caso a sinistra) abbiamo semplicemente vincolato i controlpoints della base. Non è stato applicato alcun coefficiente α o β all'oggetto, in modo da lasciarlo deformare senza che reagisse in alcun modo. E' stata scelta la modalità di visualizzazione wireframe, regolando opportunamente il tessellation, in quanto fornisce una rappresentazione più chiara della deformazione dell'oggetto nei dettagli.

Parametro	tipo	valore
massa μ	polinomiale	1.0
elasticità $\alpha_{1,1}$, $\alpha_{2,2}$,	polinomiale	0.0
rigidità $\beta_{1,1}$, $\beta_{1,2}$, $\beta_{2,2}$	polinomiale	0.0
forza in Y	polinomiale	-10.0
timestep δ_t		0.02 sec
sim. time		0.56 sec

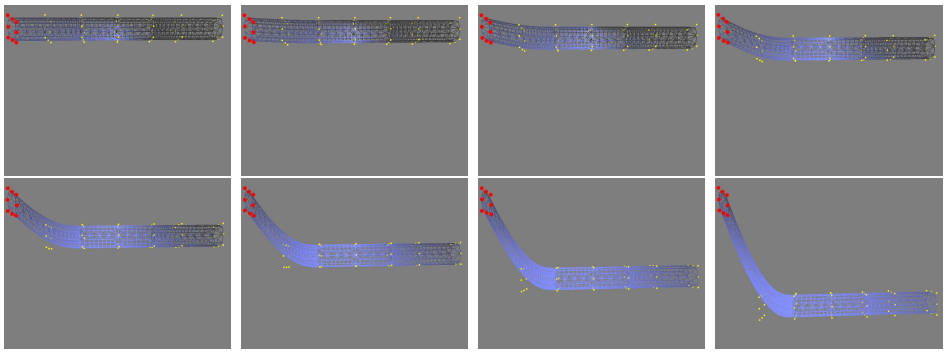

Figura 4.6: Cilindroide deformato da forza, vista ogni 0.02 sec.

Come detto in precedenza, tutti i parametri fisici sono espressi come funzioni ed è quindi possibile impostarli come tali. Nel prossimo esempio viene

fornita dimostrazione di come agisce una forza non costante, ma che varia in maniera direttamente proporzionale rispetto a v.

Parametro	tipo	grado v	valore
massa μ	polinomiale	0	1.0
elasticità $\alpha_{1,1}$	polinomiale	0	0.0
elasticità $\alpha_{2,2}$	polinomiale	0	0.0
rigidità $\beta_{1,1}$	polinomiale	0	0.000
rigidità $\beta_{1,2}$	polinomiale	0	0.000
rigidità $\beta_{2,2}$	polinomiale	0	0.000
dissipazione γ	polinomiale	0	0.0
forza in Z	polinomiale	1	$u^0 + (-10.0)u^1$
timestep δ_t			0.2 sec
sim. time			0.8 sec

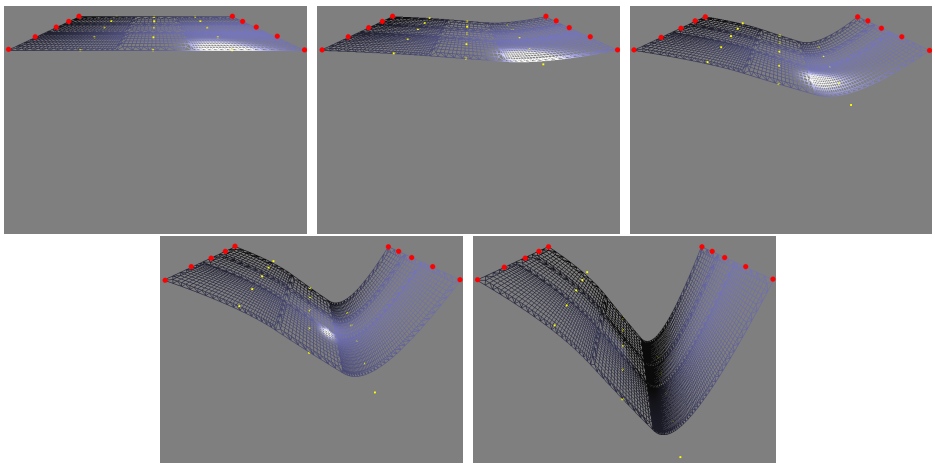

Figura 4.7: Forza non costante applicata ad un piano, visto ogni 0.2 sec.

Come è possibile notare, la forza applicata cresce proporzionalmente con v, e nella parte destra del nostro piano è superiore rispetto alla quella applicata nella parte sinistra. L'esempio fornito è molto semplice e ha come solo scopo quello di farne capire il funzionamento.

4.3.2 Forze applicate ad oggetti elastici

L'elasticità si contrappone alla forza applicata trasformando il suo potenziale elastico in energia cinetica. Come oggetto sul quale testare questa situazione è stato scelto un piano al quale sono state date specifiche caratteristiche simili ad un tappeto elastico; vincolando completamente poi i controlpoints che ne delimitano il perimetro ed applicando una forza perpendicolare al piano, si ottiene un effetto simile ad un oggetto che colpisce, con una certa forza, il nostro tappeto elastico al centro. L'elasticità è stata applicata sia al parametro $\alpha_{1,1}$ che $\alpha_{2,2}$, quindi il corpo risulta elastico sia in u che in v.

Parametro	tipo	valore
massa μ	polinomiale	1.0
elasticità $\alpha_{1,1}$,	polinomiale	1.0
elasticità $\alpha_{2,2}$,	polinomiale	1.0
rigidità $\beta_{1,2}$,	polinomiale	0.001
dissipazione γ	polinomiale	0.2
forza in Z	polinomiale	-5.0
timestep δ_t		0.2 sec
sim. time		1.4 sec

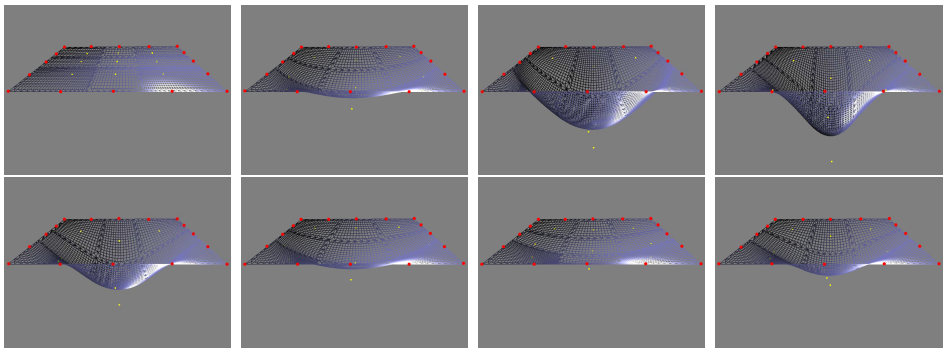

Figura 4.8: Forza applicata ad un tappeto elastico , vista ogni 0.2 sec.

Equilibrio e dissipazione

Essendo in presenza di una forza esterna applicata ad una superficie elastica, nel nostro caso un piano, dopo un tempo più o meno lungo di assestamento dovuto alle deformazioni subite, si arriverà ad una posizione di equilibrio dove il valore della forza applicata è uguale a quella che oppone il corpo elastico, mantenendolo così in una situazione di equilibrio. Viene presa in considerazione la stessa simulazione del punto precedente, ma viene analizzato un range temporale più avanzato, in modo da osservare gli ultimi stadi di assestamento e la definitiva posizione di equilibrio.

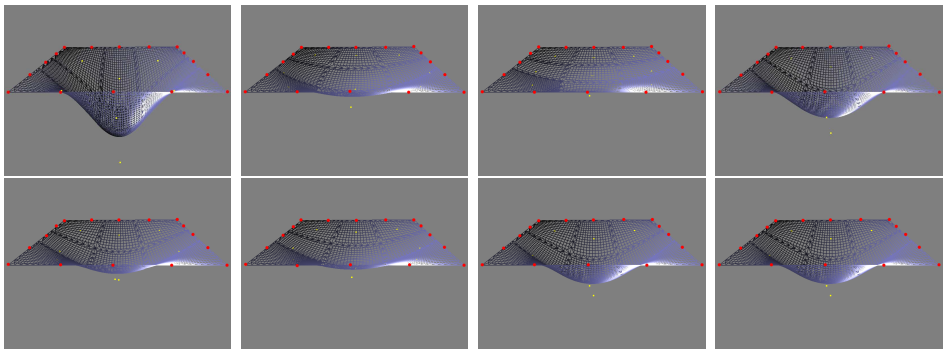

Figura 4.9: Tappeto elastico verso la posizione di equilibrio, vista ogni 0.2 sec.

Come si sarà notato, nell'esempio in questione è stato inserito anche un coefficiente diverso da 0 di dissipazione; il suo scopo è quello, ad ogni step, di disperdere una quantità stabilita di energia. Può quindi essere visto, in approssimazione, come una sorta di attrito; oltre che essere interessante dal punto di vista fisico, il suo utilizzo è stato necessario per portare l'oggetto in uno stato di equilibrio in un tempo minore, in quanto l'energia che sottrae al sistema ne limita le oscillazioni di assestamento nella fase finale, prima di trovare la stabilità.

Capitolo 5

Considerazioni

Il primo obiettivo del lavoro è stato quello di ridare piena funzionalità al software, visto i diversi anni in cui non aveva ricevuto manutenzione. Si è cercato di migliorare le parti di codice contenenti il maggior numero di fragilità, per eliminare la causa di comportamenti anomali; in un secondo momento si è deciso di migliorare, per quanto possibile, l'interfaccia di Iado per rendere più user-friendly alcuni aspetti e per migliorarne le funzionalità. Dopo aver terminato la fase "pratica" sono passato alla fase più compilativa, dove ho cerco di fornire una completa spiegazione di ogni funzionalità di Iado, sia dal punto di vista teorico (modello fisico su cui base le sue simulazioni) che dal punto di vista pratico, facendo chiarezza sull'utilizzo di tutte le parti e fornendo esempi pratici che ne mettessero in risalto le potenzialità; il tutto per dare quante più informazioni possibile ai prossimi utilizzatori. Tuttavia, come spiegato ampiamente all'interno del documento, l'intero pacchetto ha come scopo quello di fornire simulazioni pseudo-realistiche e può risultare inadeguato alla simulazione di geometrie in alcune condizioni, fornendo risultati inaspettati; i motivi principali possono essere riassunti in:

- Il numero finito di controlpoints sul quale agisce il modello fisico evidenzia subito il fatto che siamo davanti ad una approssimazione; come già citato, una geometria più ricca di controlpoints si deforma in maniera più realistica di una carente, dilatando però ovviamente i tempi di

calcolo. In Iado il compromesso realismo/cpu_time è lasciato all'utente (che può raffinare a piacimento la geometria), che può quindi decidere l'approccio desiderato.

- I vincoli meccanici danno risultati non realistici, e per questo motivo in tutte le prove condotte si è fatto sempre e solo uso di vincoli geometrici. Il motivo è da ricercare, come spiegato in [CARU98], all'acquisizione differente di massa da parte dei controlpoints, creando inaspettati risultati soprattutto in superfici chiuse

- Il calcolo ottimizzato, seppur molto vantaggioso in termini di tempo di calcolo, non dà garanzia di ottenere sempre buoni risultati, in quanto nelle operazioni di skinning non lavora sulle effettive curve sezioni ma su righe e colonne della matrice dei controlpoints. Per questo motivo non è mai stato utilizzato nelle prove sperimentali esplicate nel capitolo precedente.

Conoscendo questi limiti e la loro causa, e alla luce dei progressi effettuati a livello funzionale che hanno portato il software alla versione 1.3, l'esperienza fornita da Iado nella deformazione di oggetti è da considerarsi comunque soddisfacente e interessante anche dal punto di vista fisico.

Questi sono stati i punti cardine di questo lavoro, che però ha avuto come scopo principale quello di porre i prossimi utilizzatori nella condizione di poter sviluppare ulteriormente il software.

Possibili sviluppi futuri

- La modalità grafica di selezione dei controlpoints vincolati, pur essendo più intuitiva di quella testuale, richiede spesso un eccessivo tempo in quanto occorre selezionare uno ad uno i controlpoints stessi, cliccando col tasto destro una superficie spesso piccolissima, anche se essi sono tutti nella stessa zona e vicini tra loro. Implementare la loro selezione non più solo tramite click ma anche tramite il tracciamento di un rettangolo che

li racchiuda e li selezioni tutti quelli al suo interno contemporaneamente, farebbe risparmiare molto tempo ai futuri utilizzatori.

- Il calcolo ottimizzato, così implementato, è da considerarsi attendibile in una range limitato di situazioni e non è quindi consigliato al momento il suo utilizzo. La sua correzione, sopratutto per quanto concerne le curve chiuse, darebbe una alternativa molto più vantaggiosa in termini di tempo di calcolo alla simulazione classica.

- E' possibile migliorare ulteriormente la gestione della rotazione della geometria all'interno della finestra principale; ora come ora non è possibile ruotare la vista senza inclinarla.

- Porting del progetto su altri sistemi operativi, come MS Windows.

- Al momento Iado si appoggia a due programmi esterni, con i propri eseguibili: il primo è il motore di calcolo, *dncurv/dnsurf* e l'altro è l'implementazione della barra di avanzamento introdotta in questa versione. Entrambi hanno i loro eseguibili che vengono richiamati all'interno del codice tramite syscall. Sarebbe nettamente più pulito includere il codice sia di *dncurv/dnsurf* che di *xprogress* all'interno del del progetto Iado, rendendo la chiamata ad essi una chiamata a funzione e non una esecuzione di un programma esterno.

- L'implementazione di un sistema anche rudimentale di collisioni, per quanto consapevole della complessità che richiederebbe, renderebbe ancora più realistica la simulazione, aprendo nuovi scenari ancor più articolati.

Bibliografia

[TEQI94] D.Terzopoulos, H.Qin, Dynamic NURBS with geometric constraints for interactive sculpting, ACM Transaction on Graphics, vol.13, n.2 (1994).

[RUBI96] G.L.Rubini, Dynamic NURBS per la modellazione di superfici a forma libera, Tesi di Laurea in Informatica, terza sessione, aa 1996/97, Università di Bologna.

[CARU98] G.Casciola, G.L.Rubini,Sulle Dynamic NURBS e la simulazione di processi dinamici, Rapporto tecnico n. 1/1998 C. N. R. (1998).

[XCMODEL] G.Casciola. *xcmodel: a system to model and render NURBS curves and surfaces* User's guide (1999), `http://www.dm.unibo.it/~casciola/html/xcmodel.html`

[PIGN03] E.Pignatti, Un sistema per la modellazione dinamica mediante DNURBS ed OpenGL, Tesi di Laurea Specialistica in Informatica, seconda sessione, aa 2002/03, Università di Bologna.

[CICC03] D.Ciccone, Analisi di un pacchetto per l'animazione/deformazione di superfici, Tesi di Laurea in Informatica, terza sessione, aa 2002/03, Università di Bologna.